A Theorem Prover for Boolean BI

Jonghyun Park Jeongbong Seo Sungwoo Park

Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
Republic of Korea
{parjong,baramseo,gla}@postech.ac.kr

Abstract

While separation logic is acknowledged as an enabling technology for large-scale program verification, most of the existing verification tools use only a fragment of separation logic that excludes separating implication. As the first step towards a verification tool using full separation logic, we develop a nested sequent calculus for Boolean BI (Bunched Implications), the underlying theory of separation logic, as well as a theorem prover based on it. A salient feature of our nested sequent calculus is that its sequent may have not only smaller child sequents but also multiple parent sequents, thus producing a graph structure of sequents instead of a tree structure. Our theorem prover is based on backward search in a refinement of the nested sequent calculus in which weakening and contraction are built into all the inference rules. We explain the details of designing our theorem prover and provide empirical evidence of its practicality.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic*]: Mechanical theorem proving, Proof theory

General Terms Verification

Keywords Separation logic, Boolean BI, Theorem prover, Nested sequent calculus

1. Introduction

1.1 Separation logic

Separation logic [36] is an extension of Hoare logic which facilitates reasoning about programs using mutable data structures. As it is acknowledged as an enabling technology for large-scale program verification [7, 31, 38], researchers have developed automated verification tools that use separation logic as their foundational theory. Examples of such tools include Smallfoot [4], Space Invader [15], THOR [30], SLAyer [6], HIP [33], VeriFast [26], jStar [14], and Xisa [13]. The active development of such tools attests to the importance of local reasoning in program verification, which is precisely the key feature that separation logic intends to support.

All the aforementioned tools, however, use not full separation logic but only a decidable fragment by Berdine *et al*. [3] or its extension. Specifically separation logic features two new logical connectives, separating conjunction $\star$ and separating implication $-\!\!\star$, but this decidable fragment includes only separating conjunction. Lack of separating implication implies that for any program performing heap mutation or allocation, there is no support for backward reasoning by weakest precondition generation, an essential requirement for any complete program verification system (see Ishtiaq and O'Hearn [25]). Thus, while very effective in their respective application domains, these tools allow only forward reasoning based on symbolic execution as in [5] and fail to demonstrate the full potential of separation logic in program verification.

Omitting separating implication is not a deliberate decision. Rather it is an inevitable decision due to the availability of no theorem prover for full separation logic. Berdine *et al.* [5] suggest that such a theorem prover is highly desirable and can evolve into a complete program verification system based on separation logic:

> *This incompleteness could be dealt with if we instead used the backwards-running weakest preconditions of Separation Logic. Unfortunately, there is no existing automatic theorem prover which can deal with the form of these assertions (which use quantification and the separating implication $-\!\!\star$). If there were such a prover, we would be eager consumers of it.*

Still, however, there is no practical theorem prover for full separation logic.

Our long-term goal is to develop a theorem prover for full separation logic and incorporate it into a program verification system supporting backward reasoning. The first step is then to study Boolean BI, the underlying theory of separation logic.

1.2 Boolean BI

Boolean BI is a substructural logic which belongs to the family of the logic of BI (Bunched Implications) of O'Hearn and Pym [34]. It inherits additive connectives from classical propositional logic, thus retaining the convenience of classical reasoning. It is also particularly suitable for reasoning about local resources because of multiplicative connectives inherited from intuitionistic linear logic. Like other members in the family, Boolean BI allows us to consider free combinations of these additive connectives and multiplicative connectives, giving rise to an unusual form of contexts called bunches: trees whose internal nodes specify whether subtrees are combined additively or multiplicatively. We obtain separation logic as a model for Boolean BI based on a monoid of heaps.

While theoretical work on Boolean BI is maturing with recent discoveries of its undecidability [11, 29], there is still no practical theorem prover for Boolean BI. The display calculus for Boolean BI by Brotherston [10], which draws on the framework of display logic by Belnap [1], has the cut elimination property and thus can be easily turned into a theorem prover, but developing a practical proof search strategy on top of it does not seem to be easy because of the complexity due to its display rules [9]. In order to develop

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

POPL'13, January 23–25, 2013, Rome, Italy.

Copyright © 2013 ACM 978-1-4503-1832-7/13/01. . . $10.00

a practical theorem prover for Boolean BI and hence also for full separation logic, we choose to develop another proof theory that directly reflects the characteristics of Boolean BI and lends itself well to proof search. This paper presents such a proof theory for Boolean BI as well as a theorem prover based on it.

1.3 Contribution

We present a *nested sequent calculus* $\mathbf{S_{BBI}}$ for Boolean BI. Unlike in typical nested sequent calculi [12, 22–24, 27], its sequent may have not only smaller child sequents but also multiple parent sequents, thus producing a graph structure of sequents instead of a tree structure. The use of nested sequents is necessary because of the presence of intuitionistic multiplicative conjunction in a classical setting. The use of a graph structure of sequents is necessary because of the interaction between multiplicative implication and classical negation. As in typical multi-conclusioned sequent calculi for classical logic, we use multisets of formulas not only for antecedents but also for succedents of a sequent. Thus sequents in $\mathbf{S_{BBI}}$ do not use bunches, which are supplanted by new structural connectives specifying a graph structure of sequents. $\mathbf{S_{BBI}}$ has the cut elimination property and is sound and complete with respect to the Kripke semantics for Boolean BI.

Our theorem prover for Boolean BI is based on backward proof search in another nested sequent calculus $\mathbf{CS_{BBI}}$ which is obtained from $\mathbf{S_{BBI}}$ by building weakening and contraction into all the inference rules. In conjunction with a graph structure of sequents, the structural rules in $\mathbf{CS_{BBI}}$ make it particularly challenging to devise a practical proof search strategy, even if it is based on backward proof search. We deal with an explosion in the search space due to the structural rules, which can be applied indefinitely and exponentially increase the search space, by prioritizing their applications. We find that our theorem prover is reasonably fast in proving typical formulas of Boolean BI. To the best of our knowledge, our theorem prover is the first theorem prover for Boolean BI.

1.4 Organization of the paper

Section 2 gives preliminaries on Boolean BI. Section 3 presents the nested sequent calculus $\mathbf{S_{BBI}}$ and the satisfaction relation for its sequents. Section 4 proves the cut elimination property of $\mathbf{S_{BBI}}$, and Section 5 proves the soundness and completeness of $\mathbf{S_{BBI}}$ with respect to the satisfaction relation as well as the Kripke semantics for Boolean BI. Section 6 reviews the display calculus for Boolean BI by Brotherston [10] and shows that $\mathbf{S_{BBI}}$ is an optimization of the display calculus. Section 7 presents the nested sequent calculus $\mathbf{CS_{BBI}}$. Section 8 describes the backward search strategy in our theorem prover and presents experimental results. Section 9 discusses related work and Section 10 concludes. Our theorem prover (with an online demo) and accompanying technical report are available at `http://pl.postech.ac.kr/BBI/`.

2. Preliminaries on Boolean BI

Formulas in Boolean BI extend classical propositional logic with multiplicative connectives from linear logic:

$$\text{formula}\quad A \quad ::= \quad P \mid \bot \mid \neg A \mid A \vee A \mid \mathsf{I} \mid A \star A \mid A \mathrel{-\!\!\star} A$$

P denotes an atomic formula drawn from a set V. I is the multiplicative unit. $A \star B$ is a multiplicative conjunction and $A \mathrel{-\!\!\star} B$ is a multiplicative implication. We define $\top$ as $\neg\bot$, $A \wedge B$ as $\neg(\neg A \vee \neg B)$, and $A \to B$ as $\neg A \vee B$. We use conventional precedence rules for logical connectives: $\neg > \wedge, \star > \vee > \to, \mathrel{-\!\!\star}$.

The Kripke semantics of Boolean BI [17] uses a non-deterministic commutative monoid on a set U. Assume a binary operator

$$\begin{array}{rll}
w, \rho \models P & \text{iff.} & w \in \rho(P) \\
w, \rho \models \bot & \text{iff.} & \text{never} \\
w, \rho \models \neg A & \text{iff.} & w, \rho \not\models A \\
w, \rho \models A \vee B & \text{iff.} & w, \rho \models A \text{ or } w, \rho \models B \\
w, \rho \models \mathsf{I} & \text{iff.} & w = e \\
w, \rho \models A \star B & \text{iff.} & \exists w_1, w_2 \in U \text{ such that } w \in w_1 \circ w_2 \\
 & & \text{and } w_1, \rho \models A \text{ and } w_2, \rho \models B \\
w, \rho \models A \mathrel{-\!\!\star} B & \text{iff.} & \forall w_1 \in U.\ w_1, \rho \models A \text{ implies} \\
 & & \forall w_2 \in w \circ w_1.\ w_2, \rho \models B
\end{array}$$

Figure 1. Satisfaction relation $w, \rho \models A$ for formulas

$$\begin{array}{ll}
\text{(Axiom 1)} & \vdash A \to \mathsf{I} \star A \\
\text{(Axiom 2)} & \vdash \mathsf{I} \star A \to A \\
\text{(Axiom 3)} & \vdash A \star B \to B \star A \\
\text{(Axiom 4)} & \vdash A \star (B \star C) \to (A \star B) \star C
\end{array}$$

$$\frac{\vdash A_1 \to A_2 \quad \vdash B_1 \to B_2}{\vdash (A_1 \star B_1) \to (A_2 \star B_2)}\ \star H$$

$$\frac{\vdash (A \star B) \to C}{\vdash A \to (B \mathrel{-\!\!\star} C)}\ \mathrel{-\!\!\star} H_1 \qquad \frac{\vdash A \to (B \mathrel{-\!\!\star} C)}{\vdash (A \star B) \to C}\ \mathrel{-\!\!\star} H_2$$

Figure 2. Axioms and inference rules for multiplicative connectives in the Hilbert system for Boolean BI

$\circ : U \times U \to \mathcal{P}(U)$ and a unit element $e \in U$ (where $\mathcal{P}(U)$ denotes the power set of U). We extend $\circ$ to a binary operator on $\mathcal{P}(U)$ such that $U_1 \circ U_2 = \cup\{w_1 \circ w_2 \mid w_1 \in U_1, w_2 \in U_2\}$. A non-deterministic commutative monoid is a triple $\langle U, \circ, e\rangle$ which satisfies the following conditions:

$$\begin{array}{rl}
\text{(neutrality)} & \forall w \in U.\ w \circ e = \{w\} \\
\text{(commutativity)} & \forall w_1, w_2 \in U.\ w_1 \circ w_2 = w_2 \circ w_1 \\
\text{(associativity)} & \forall w_1, w_2, w_3 \in U.\ w_1 \circ (w_2 \circ w_3) = \\
 & \qquad (w_1 \circ w_2) \circ w_3
\end{array}$$

Given a non-deterministic commutative monoid $\langle U, \circ, e\rangle$ and a valuation $\rho : V \to \mathcal{P}(U)$ of atomic formulas, we obtain the Kripke semantics of Boolean BI from the satisfaction relation $w, \rho \models A$ for formulas given in Figure 1. The satisfaction relation $w, \rho \models A$ is defined inductively on the structure of formula A. A formula A is valid, written $\models A$, if $w, \rho \models A$ holds for any element w and valuation ρ.

The Hilbert system for Boolean BI [35] uses a judgment $\vdash A$ and is obtained by extending classical propositional logic with axioms and inference rules for multiplicative connectives given in Figure 2. An induction on the structure of the proof of $\vdash A$ proves the soundness of the Hilbert system with respect to the Kripke semantics of Boolean BI. Galmiche and Larchey-Wendling [17] prove that the Hilbert system is also complete with respect to the Kripke semantics of Boolean BI.

Theorem 2.1. *$\models A$ if and only if $\vdash A$.*

3. Nested sequent calculus $\mathbf{S_{BBI}}$ for Boolean BI

This section presents the nested sequent calculus $\mathbf{S_{BBI}}$ for Boolean BI. We first explain the definition of sequents in $\mathbf{S_{BBI}}$. Then we present the satisfaction relation for sequents and the inference rules of $\mathbf{S_{BBI}}$.

3.1 Nested sequents

A sequent in $\mathbf{S_{BBI}}$ represents a graph structure whose nodes store sequents in classical logic. A node can have multiple parent nodes

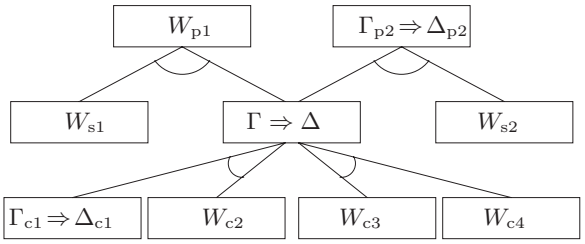


Figure 3. An example of a graph structure of nodes in $\mathbf{S_{BBI}}$ (p for parent, c for child, s for sibling)

as well as multiple child nodes, but the following two relations should always hold:

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.

Formally a sequent W describes a graph structure with respect to a certain node in it, which we refer to as the *reference node.* It consists of a truth context and a falsehood context. A truth context contains *node states* which are either true formulas specific to the reference node or descriptions of its relation to child, sibling, and parent nodes; a falsehood context contains false formulas specific to the reference node:

$$\begin{array}{rlll} \text{sequent} & W & = & \Gamma \Rightarrow \Delta \\ \text{truth context} & \Gamma & ::= & \cdot \mid \Gamma; S \\ \text{falsehood context} & \Delta & ::= & \cdot \mid \Delta; A \\ \text{node state} & S & ::= & A \mid \emptyset_{\mathsf{m}} \mid W, W \mid W \langle\!\langle W \rangle\!\rangle \end{array}$$

We use truth contexts and falsehood contexts as unordered sets and do not use an additive zero (like $\emptyset_{\mathsf{a}}$ in the definition of bunches). $\emptyset_{\mathsf{m}}$ is a special node state which corresponds to the unit element of the monoid in the Kripke semantics of Boolean BI. A *multiplicative pair* W, W' asserts the existence of a pair of child nodes which are reference nodes of W and W'. An *adjoint pair* $W \langle\!\langle W' \rangle\!\rangle$ asserts the existence of a sibling node and a common parent node which are reference nodes of W and W', respectively. Note that a sequent in $\mathbf{S_{BBI}}$ reverts to a sequent in classical logic if we leave only true formulas in its truth context.

The use of adjoint pairs implies that we can describe the same graph structure of nodes using different sequents by changing the reference node. As an example, consider the graph structure in Figure 3 where lines denote parent-child relations and arcs denote sibling relations. We let $\Gamma' = W_{\mathrm{s1}} \langle\!\langle W_{\mathrm{p1}} \rangle\!\rangle; (W_{\mathrm{c3}}, W_{\mathrm{c4}})$. Then the following three sequents describe the same graph structure in Figure 3, but all use different reference nodes (top right, center, and bottom left):

$$\begin{array}{l} \Gamma_{\mathrm{p2}}; (\Gamma; \Gamma'; (\Gamma_{\mathrm{c1}} \Rightarrow \Delta_{\mathrm{c1}}, W_{\mathrm{c2}}) \Rightarrow \Delta, W_{\mathrm{s2}}) \Rightarrow \Delta_{\mathrm{p2}} \\ \Gamma; \Gamma'; (\Gamma_{\mathrm{c1}} \Rightarrow \Delta_{\mathrm{c1}}, W_{\mathrm{c2}}); W_{\mathrm{s2}} \langle\!\langle \Gamma_{\mathrm{p2}} \Rightarrow \Delta_{\mathrm{p2}} \rangle\!\rangle \Rightarrow \Delta \\ \Gamma_{\mathrm{c1}}; W_{\mathrm{c2}} \langle\!\langle \Gamma; \Gamma'; W_{\mathrm{s2}} \langle\!\langle \Gamma_{\mathrm{p2}} \Rightarrow \Delta_{\mathrm{p2}} \rangle\!\rangle \Rightarrow \Delta \rangle\!\rangle \Rightarrow \Delta_{\mathrm{c1}} \end{array}$$

$\mathbf{S_{BBI}}$ provides two inference rules which convert a sequent into another equivalent sequent by changing the reference node.

Our definition of sequents in $\mathbf{S_{BBI}}$ embodies the principle of proof by contradiction from classical logic: a proof of a sequent means that its truth and falsehood contexts together lead to a logical contradiction. This departure from the standard interpretation of sequents for classical logic (in which the conjunction of antecedents implies the disjunction of succedents) is intentional, as the principle of proof by contradiction guides the development of both the satisfaction relation and the rules for $\mathbf{S_{BBI}}$.

3.2 Satisfaction relation for sequents

Given a non-deterministic commutative monoid $\langle U, \circ, e \rangle$ and a valuation $\rho : V \to \mathcal{P}(U)$ of atomic formulas, we can define the satisfaction relation $w, \rho \models_{\mathcal{W}} W$ for sequents. It uses another satisfaction relation $w, \rho \models_{\mathcal{S}} S$ for node states and the satisfaction relation $w, \rho \models A$ for formulas:

$$w, \rho \models_{\mathcal{W}} \Gamma \Rightarrow \Delta \quad \text{iff.} \quad \begin{cases} \forall S \in \Gamma.\ w, \rho \models_{\mathcal{S}} S \\ \forall A \in \Delta.\ w, \rho \not\models A \end{cases}$$

The satisfaction relation $w, \rho \models_{\mathcal{S}} S$ for node states is defined as follows:

$$\begin{array}{rll} w, \rho \models_{\mathcal{S}} A & \text{iff.} & w, \rho \models A \\ w, \rho \models_{\mathcal{S}} \emptyset_{\mathsf{m}} & \text{iff.} & w = e \\ w, \rho \models_{\mathcal{S}} W_1, W_2 & \text{iff.} & \exists w_1, w_2 \in U \text{ such that } w \in w_1 \circ w_2 \\ & & \text{and } w_1, \rho \models_{\mathcal{W}} W_1 \text{ and } w_2, \rho \models_{\mathcal{W}} W_2 \\ w, \rho \models_{\mathcal{S}} W_1 \langle\!\langle W_2 \rangle\!\rangle & \text{iff.} & \exists w_1, w_2 \in U \text{ such that } w_2 \in w \circ w_1 \\ & & \text{and } w_1, \rho \models_{\mathcal{W}} W_1 \text{ and } w_2, \rho \models_{\mathcal{W}} W_2 \end{array}$$

Note that the satisfaction relation for multiplicative formulas can be rewritten in terms of the satisfaction relation for node states as follows:

- $w, \rho \models \mathsf{I}$ iff. $w, \rho \models_{\mathcal{S}} \emptyset_{\mathsf{m}}$.
- $w, \rho \models A \star B$ iff. $w, \rho \models_{\mathcal{S}} (A \Rightarrow \cdot), (B \Rightarrow \cdot)$.
- $w, \rho \models A \mathrel{-\!\!\star} B$ iff. $w, \rho \not\models_{\mathcal{S}} (A \Rightarrow \cdot) \langle\!\langle \cdot \Rightarrow B \rangle\!\rangle$.

If $w, \rho \not\models_{\mathcal{W}} W$ holds for any element w and valuation ρ, we say that W is unsatisfiable and write $\not\models_{\mathcal{W}} W$.

3.3 Nested sequent calculus $\mathbf{S_{BBI}}$

Figure 4 shows the nested sequent calculus $\mathbf{S_{BBI}}$ for Boolean BI. The inference rules are divided into three groups: structural rules, traverse rules, and logical rules. We read every rule from the conclusion to the premise.

A structural rule makes a change to the sequent in the conclusion, but does not change the reference node. The rules $WL_{\mathcal{S}}$, $WR_{\mathcal{S}}$, $CL_{\mathcal{S}}$, and $CR_{\mathcal{S}}$ are weakening and contraction rules. The rules $EC_{\mathcal{S}}$ and $EA_{\mathcal{S}}$ rewrite a node state according to commutativity and associativity of sequents, respectively. Note that associativity of sequents does not use $(W_1, W_2), W_3$ and $W_1, (W_2, W_3)$, both of which are syntactically ill-formed. The rule $\emptyset_{\mathsf{m}} U_{\mathcal{S}}$ creates a new child node with a special form of sequent $\emptyset_{\mathsf{m}} \Rightarrow \cdot$, which can be absorbed back into the parent node by the rule $\emptyset_{\mathsf{m}} D_{\mathcal{S}}$. Intuitively $\emptyset_{\mathsf{m}} \Rightarrow \cdot$ describes an empty node whose sibling node can be identified with its parent node.

A traverse rule changes the reference node without changing parent-child or sibling relations between nodes. The rules $TC_{\mathcal{S}}$ and $TP_{\mathcal{S}}$ promote the left child node (corresponding to $\Gamma_{\mathrm{c1}} \Rightarrow \Delta_{\mathrm{c1}}$) and the parent node (corresponding to $\Gamma_{\mathrm{p}} \Rightarrow \Delta_{\mathrm{p}}$), respectively, as the new reference node. In conjunction with the rule $EC_{\mathcal{S}}$, the two traverse rules enable us to designate an arbitrary node as the reference node because every pair of nodes can be connected only via parent-child relations. The following example shows how to promote the sibling node as the reference node:

$$\dfrac{\dfrac{\dfrac{\Gamma_{\mathrm{s}}; (\Gamma \Rightarrow \Delta) \langle\!\langle \Gamma_{\mathrm{p}} \Rightarrow \Delta_{\mathrm{p}} \rangle\!\rangle \Rightarrow \Delta_{\mathrm{s}}}{\Gamma_{\mathrm{p}}; (\Gamma_{\mathrm{s}} \Rightarrow \Delta_{\mathrm{s}}), (\Gamma \Rightarrow \Delta) \Rightarrow \Delta_{\mathrm{p}}}\, TC_{\mathcal{S}}}{\Gamma_{\mathrm{p}}; (\Gamma \Rightarrow \Delta), (\Gamma_{\mathrm{s}} \Rightarrow \Delta_{\mathrm{s}}) \Rightarrow \Delta_{\mathrm{p}}}\, EC_{\mathcal{S}}}{\Gamma; (\Gamma_{\mathrm{s}} \Rightarrow \Delta_{\mathrm{s}}) \langle\!\langle \Gamma_{\mathrm{p}} \Rightarrow \Delta_{\mathrm{p}} \rangle\!\rangle \Rightarrow \Delta}\, TP_{\mathcal{S}}$$

A logical rule focuses on a principal formula in the reference node. If the sequent already expresses a logical contradiction, it

Structural rules:

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma; S \Rightarrow \Delta}\ WL_{\mathcal{S}} \qquad \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta; A}\ WR_{\mathcal{S}} \qquad \frac{\Gamma; S; S \Rightarrow \Delta}{\Gamma; S \Rightarrow \Delta}\ CL_{\mathcal{S}} \qquad \frac{\Gamma \Rightarrow \Delta; A; A}{\Gamma \Rightarrow \Delta; A}\ CR_{\mathcal{S}} \qquad \frac{\Gamma; W', W \Rightarrow \Delta}{\Gamma; W, W' \Rightarrow \Delta}\ EC_{\mathcal{S}}$$

$$\frac{\Gamma; W_1, (W_2, W_3 \Rightarrow \cdot) \Rightarrow \Delta}{\Gamma; (W_1, W_2 \Rightarrow \cdot), W_3 \Rightarrow \Delta}\ EA_{\mathcal{S}} \qquad \frac{\Gamma_1; (\Gamma_2 \Rightarrow \Delta_2), (\emptyset_{\mathsf{m}} \Rightarrow \cdot) \Rightarrow \Delta_1}{\Gamma_1; \Gamma_2 \Rightarrow \Delta_1; \Delta_2}\ \emptyset_{\mathsf{m}} U_{\mathcal{S}} \qquad \frac{\Gamma_1; \Gamma_2 \Rightarrow \Delta_1; \Delta_2}{\Gamma_1; (\Gamma_2 \Rightarrow \Delta_2), (\emptyset_{\mathsf{m}} \Rightarrow \cdot) \Rightarrow \Delta_1}\ \emptyset_{\mathsf{m}} D_{\mathcal{S}}$$

Traverse rules:

$$\frac{\Gamma_{\mathrm{c}1}; (\Gamma_{\mathrm{c}2} \Rightarrow \Delta_{\mathrm{c}2}) \langle\!\langle \Gamma \Rightarrow \Delta \rangle\!\rangle \Rightarrow \Delta_{\mathrm{c}1}}{\Gamma; (\Gamma_{\mathrm{c}1} \Rightarrow \Delta_{\mathrm{c}1}), (\Gamma_{\mathrm{c}2} \Rightarrow \Delta_{\mathrm{c}2}) \Rightarrow \Delta}\ TC_{\mathcal{S}} \qquad \frac{\Gamma_{\mathrm{p}}; (\Gamma \Rightarrow \Delta), (\Gamma_{\mathrm{s}} \Rightarrow \Delta_{\mathrm{s}}) \Rightarrow \Delta_{\mathrm{p}}}{\Gamma; (\Gamma_{\mathrm{s}} \Rightarrow \Delta_{\mathrm{s}}) \langle\!\langle \Gamma_{\mathrm{p}} \Rightarrow \Delta_{\mathrm{p}} \rangle\!\rangle \Rightarrow \Delta}\ TP_{\mathcal{S}}$$

(p for parent, c for child, s for sibling)

Logical rules:

$$\frac{}{A \Rightarrow A}\ Init_{\mathcal{S}} \qquad \frac{}{\bot \Rightarrow \cdot}\ \bot L_{\mathcal{S}} \qquad \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta; \bot}\ \bot R_{\mathcal{S}} \qquad \frac{\Gamma \Rightarrow \Delta; A}{\Gamma; \neg A \Rightarrow \Delta}\ \neg L_{\mathcal{S}} \qquad \frac{\Gamma; A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta; \neg A}\ \neg R_{\mathcal{S}} \qquad \frac{\Gamma_1; A \Rightarrow \Delta_1 \quad \Gamma_2; B \Rightarrow \Delta_2}{\Gamma_1; \Gamma_2; A \vee B \Rightarrow \Delta_1; \Delta_2}\ \vee L_{\mathcal{S}}$$

$$\frac{\Gamma \Rightarrow \Delta; A; B}{\Gamma \Rightarrow \Delta; A \vee B}\ \vee R_{\mathcal{S}} \qquad \frac{\Gamma; \emptyset_{\mathsf{m}} \Rightarrow \Delta}{\Gamma; \mathsf{I} \Rightarrow \Delta}\ \mathsf{I} L_{\mathcal{S}} \qquad \frac{}{\emptyset_{\mathsf{m}} \Rightarrow \mathsf{I}}\ \mathsf{I} R_{\mathcal{S}} \qquad \frac{\Gamma; (A \Rightarrow \cdot), (B \Rightarrow \cdot) \Rightarrow \Delta}{\Gamma; A \star B \Rightarrow \Delta}\ \star L_{\mathcal{S}}$$

$$\frac{\Gamma_1 \Rightarrow \Delta_1; A \quad \Gamma_2 \Rightarrow \Delta_2; B}{(\Gamma_1 \Rightarrow \Delta_1), (\Gamma_2 \Rightarrow \Delta_2) \Rightarrow A \star B}\ \star R_{\mathcal{S}} \qquad \frac{\Gamma_1 \Rightarrow \Delta_1; A \quad \Gamma_2; B \Rightarrow \Delta_2}{(\Gamma_1 \Rightarrow \Delta_1) \langle\!\langle \Gamma_2 \Rightarrow \Delta_2 \rangle\!\rangle; A \mathrel{-\!\!\star} B \Rightarrow \cdot}\ \mathrel{-\!\!\star} L_{\mathcal{S}} \qquad \frac{\Gamma; (A \Rightarrow \cdot) \langle\!\langle \cdot \Rightarrow B \rangle\!\rangle \Rightarrow \Delta}{\Gamma \Rightarrow \Delta; A \mathrel{-\!\!\star} B}\ \mathrel{-\!\!\star} R_{\mathcal{S}}$$

Figure 4. Nested sequent calculus $\mathbf{S_{BBI}}$ for Boolean BI

completes the proof without generating a premise. All the rules from $Init_{\mathcal{S}}$ to $\vee R_{\mathcal{S}}$ are from classical propositional logic. The rule $Init_{\mathcal{S}}$ expresses the principle of proof by contradiction. The rules $\mathsf{I}L_{\mathcal{S}}$ and $\mathsf{I}R_{\mathcal{S}}$ use the fact that I is true only in an empty node which $\emptyset_{\mathsf{m}}$ describes. The rules $\star L_{\mathcal{S}}$ and $\star R_{\mathcal{S}}$ are based on the following interpretation of multiplicative conjunction $\star$:

$$\begin{array}{lll} w, \rho \models A \star B & \text{iff.} & \exists w_1, w_2 \in U \text{ such that } w \in w_1 \circ w_2 \\ & & \text{and } w_1, \rho \models A \text{ and } w_2, \rho \models B \\ w, \rho \not\models A \star B & \text{iff.} & \forall w_1, w_2 \in U.\ w \in w_1 \circ w_2 \text{ implies} \\ & & w_1, \rho \not\models A \text{ or } w_2, \rho \not\models B \end{array}$$

The rule $\star L_{\mathcal{S}}$ creates ($\exists$) two fresh child nodes (corresponding to w_1 and w_2) where A and B are true, respectively, which explains why we need to use nested sequents. The rule $\star R_{\mathcal{S}}$ chooses ($\forall$) two existing child nodes (corresponding to w_1 and w_2) which are described by $\Gamma_1 \Rightarrow \Delta_1$ and $\Gamma_2 \Rightarrow \Delta_2$. The rules $\mathrel{-\!\!\star} L_{\mathcal{S}}$ and $\mathrel{-\!\!\star} R_{\mathcal{S}}$ are based on the following interpretation of multiplicative implication $\mathrel{-\!\!\star}$:

$$\begin{array}{lll} w, \rho \models A \mathrel{-\!\!\star} B & \text{iff.} & \forall w_1 \in U.\ w_1, \rho \models A \text{ implies} \\ & & \forall w_2 \in w \circ w_1.\ w_2, \rho \models B \\ w, \rho \not\models A \mathrel{-\!\!\star} B & \text{iff.} & \exists w_1, w_2 \in U \text{ such that } w_2 \in w \circ w_1 \\ & & \text{and } w_1, \rho \models A \text{ and } w_2, \rho \not\models B \end{array}$$

The rule $\mathrel{-\!\!\star} L_{\mathcal{S}}$ chooses ($\forall$) existing sibling and parent nodes (corresponding to w_1 and w_2) which are described by $\Gamma_1 \Rightarrow \Delta_1$ and $\Gamma_2 \Rightarrow \Delta_2$. The rule $\mathrel{-\!\!\star} R_{\mathcal{S}}$ creates ($\exists$) a fresh sibling node (corresponding to w_1) where A is true and a fresh parent node (corresponding to w_2) where B is false, which explains why we need to allow multiple parent nodes.

Figure 5 shows an example of proving $A \to (A \star B) \vee (A \star \neg B)$ in $\mathbf{S_{BBI}}$. The formula means that every node can have an adjacent node in which either B or $\neg B$ is true. First we apply the rule $\emptyset_{\mathsf{m}} U_{\mathcal{S}}$ to create an empty node, described by $\emptyset_{\mathsf{m}} \Rightarrow \cdot$, in which we later mix assumptions of B and $\neg B$ to produce a logical contradiction:

$$(A \Rightarrow \cdot), (\emptyset_{\mathsf{m}} \Rightarrow \cdot) \Rightarrow A \star B; A \star \neg B$$

Then we extend the truth context of the empty node with a node state S describing the current relation with its sibling and parent nodes:

$$(A \Rightarrow \cdot), (\emptyset_{\mathsf{m}}; S \Rightarrow \cdot) \Rightarrow A \star B; A \star \neg B$$

Here we promote the empty node as the reference node to generate S and apply the contraction rule $CL_{\mathcal{S}}$ to duplicate S. After isolating the sequent for the empty node and adding B to its falsehood context, we consume S in $\emptyset_{\mathsf{m}}; S \Rightarrow B$ (by the rule $TP_{\mathcal{S}}$) to restore

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \Rightarrow A}\ Init_{\mathcal{S}}{}^{8}}{(A \Rightarrow \cdot), (\emptyset_{\mathsf{m}}; S \Rightarrow \cdot) \Rightarrow A \star B}\ \star R_{\mathcal{S}}{}^{7}}{(A \Rightarrow \cdot), (\emptyset_{\mathsf{m}}; S \Rightarrow \cdot) \Rightarrow A \star B; A \star \neg B}\ WR_{\mathcal{S}}{}^{6}}{(\emptyset_{\mathsf{m}}; S \Rightarrow \cdot), (A \Rightarrow \cdot) \Rightarrow A \star B; A \star \neg B}\ \left.\begin{array}{l} EC_{\mathcal{S}} \\ TP_{\mathcal{S}} \end{array}\right\}{}^{5}}{\dfrac{\emptyset_{\mathsf{m}}; S; S \Rightarrow \cdot}{\emptyset_{\mathsf{m}}; S \Rightarrow \cdot}\ CL_{\mathcal{S}}{}^{4}}}{(\emptyset_{\mathsf{m}} \Rightarrow \cdot), (A \Rightarrow \cdot) \Rightarrow A \star B; A \star \neg B}\ \left.\begin{array}{l} TC_{\mathcal{S}} \\ EC_{\mathcal{S}} \end{array}\right\}{}^{3}}{(A \Rightarrow \cdot), (\emptyset_{\mathsf{m}} \Rightarrow \cdot) \Rightarrow A \star B; A \star \neg B}\ \emptyset_{\mathsf{m}} U_{\mathcal{S}}{}^{2}}{A \Rightarrow A \star B; A \star \neg B}}{A \Rightarrow (A \star B) \vee (A \star \neg B)}\ \vee R_{\mathcal{S}}{}^{1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B \Rightarrow B}\ Init_{\mathcal{S}}{}^{15}}{\emptyset_{\mathsf{m}}; B \Rightarrow B}\ WL_{\mathcal{S}}{}^{14}}{\emptyset_{\mathsf{m}} \Rightarrow B; \neg B}\ \neg R_{\mathcal{S}}{}^{13}}{(A \Rightarrow \cdot), (\emptyset_{\mathsf{m}} \Rightarrow B) \Rightarrow A \star \neg B}\ \star R_{\mathcal{S}}{}^{11}}{(A \Rightarrow \cdot), (\emptyset_{\mathsf{m}} \Rightarrow B) \Rightarrow A \star B; A \star \neg B}\ WR_{\mathcal{S}}{}^{10}}{(\emptyset_{\mathsf{m}} \Rightarrow B), (A \Rightarrow \cdot) \Rightarrow A \star B; A \star \neg B}\ \left.\begin{array}{l} EC_{\mathcal{S}} \\ TP_{\mathcal{S}} \end{array}\right\}{}^{9}}{\emptyset_{\mathsf{m}}; S \Rightarrow B}$$

where $S = (A \Rightarrow \cdot) \langle\!\langle \cdot \Rightarrow A \star B; A \star \neg B \rangle\!\rangle$

Figure 5. A proof of $A \Rightarrow (A \star B) \vee (A \star \neg B)$ in $\mathbf{S_{BBI}}$. We number all proof steps for comparison with Figure 6.

the previous relation between the empty node and its sibling and parent nodes:

$$(A \Rightarrow \cdot), (\emptyset_{\mathsf{m}} \Rightarrow B) \Rightarrow A \star B; A \star \neg B$$

Finally we add B to the truth context and produce a logical contradiction:

$$\emptyset_{\mathsf{m}}; B \Rightarrow B$$

4. Cut elimination in $\mathbf{S_{BBI}}$

We state the cut elimination property of $\mathbf{S_{BBI}}$ as follows:[1]

Theorem 4.1 (Cut elimination)**.**
If $\Gamma \Rightarrow \Delta; C$ *and* $\Gamma'; C \Rightarrow \Delta'$*, then* $\Gamma; \Gamma' \Rightarrow \Delta; \Delta'$.

[1] Strictly speaking, Theorem 4.1 states only the admissibility of the cut rule. The cut elimination property, however, immediately follows as a corollary and we refer to Theorem 4.1 as the cut elimination theorem.

Section 6 gives an indirect proof of Theorem 4.1 which exploits the cut elimination property of the display calculus for Boolean BI [10]. Here we give a sketch of a direct proof which is inspired by the proof of cut elimination in original display logic [1].

The main complication in proving Theorem 4.1 is that the two contraction rules $CL_{\mathcal{S}}$ and $CR_{\mathcal{S}}$ duplicate a node state or a formula in their premise. In conjunction with the traverse rules, these contraction rules can produce copies of the cut formula C in different (smaller) sequents within the same sequent, as in:

$$\Gamma; (\Gamma_1; C^{n_1} \Rightarrow \Delta_1), W_1; (\Gamma_2; C^{n_2} \Rightarrow \Delta_2), W_2 \Rightarrow \Delta$$

Here C^n means a truth context containing n copies of C. To represent such a sequent containing smaller sequents with copies of the cut formula, we introduce the following definitions:

$$\begin{array}{rcll} \text{partial sequent} & \omega & ::= & [] \mid [; \gamma \Rightarrow] \mid \Gamma; \gamma \Rightarrow \Delta \\ \text{partial truth context} & \gamma & ::= & \sigma \mid \sigma; \gamma \\ \text{partial node state} & \sigma & ::= & \omega, W \mid W, \omega \mid \omega, \omega \mid \\ & & & \omega\langle\!\langle W \rangle\!\rangle \mid W\langle\!\langle \omega \rangle\!\rangle \mid \omega\langle\!\langle \omega \rangle\!\rangle \end{array}$$

A partial sequent is a sequent with one or more holes in it; similarly partial truth contexts and partial node states contain one or more holes. We write $\omega[W_1] \cdots [W_n]$ for the sequent obtained by filling holes in ω with sequents $W_1, \cdots, W_n$ in that order; we define $\gamma[W_1] \cdots [W_n]$ and $\sigma[W_1] \cdots [W_n]$ in a similar way:

$$\omega[W_1] \cdots [W_n] = \begin{cases} W_1 & \text{where } \omega = [], n = 1 \\ \Gamma_1; \gamma[W_2] \cdots [W_n] \Rightarrow \Delta_1 & \text{where } \omega = [; \gamma \Rightarrow], \\ & \quad n > 1, W_1 = \Gamma_1 \Rightarrow \Delta_1 \\ \Gamma; \gamma[W_1] \cdots [W_n] \Rightarrow \Delta & \text{where } \omega = \Gamma; \gamma \Rightarrow \Delta \end{cases}$$

Note that $[; \gamma \Rightarrow]$ uses the first sequent to describe the reference node and remaining sequents to fill the holes in γ.

The proof of Theorem 4.1, which is inspired by the proof of cut elimination in display logic [1], proceeds by proving the following three lemmas. Here we say that $/C/$ holds if $\Gamma \Rightarrow \Delta; A$ and $\Gamma'; A \Rightarrow \Delta'$ implies $\Gamma; \Gamma' \Rightarrow \Delta; \Delta'$ for any proper subformula A of C. We also write $\Gamma; \boxed{C} \Rightarrow \Delta$ and $\Gamma \Rightarrow \Delta; \boxed{C}$ to indicate that C is the principal formula of the last inference rule in their proofs. The proof of Lemma 4.3 uses Lemma 4.2, and the proof of Lemma 4.4 uses Lemma 4.3.

Lemma 4.2. *Suppose that $/C/$ holds. Then*
$\Gamma \Rightarrow \Delta; \boxed{C}$ *and* $\Gamma'; \boxed{C} \Rightarrow \Delta'$ *imply* $\Gamma; \Gamma' \Rightarrow \Delta; \Delta'$.

Lemma 4.3. *Suppose that $/C/$ holds. Then*
$\omega[\Gamma_1 \Rightarrow \Delta_1; C^{n_1}] \cdots [\Gamma_k \Rightarrow \Delta_k; C^{n_k}]$ *and* $\Gamma'; \boxed{C} \Rightarrow \Delta'$
imply $\omega[\Gamma_1; \Gamma' \Rightarrow \Delta_1; \Delta'] \cdots [\Gamma_k; \Gamma' \Rightarrow \Delta_k; \Delta']$.

Lemma 4.4. *Suppose that $/C/$ holds. Then*
$\Gamma \Rightarrow \Delta; C$ *and* $\omega[\Gamma'_1; C^{n_1} \Rightarrow \Delta'_1] \cdots [\Gamma'_k; C^{n_k} \Rightarrow \Delta'_k]$
imply $\omega[\Gamma; \Gamma'_1 \Rightarrow \Delta; \Delta'_1] \cdots [\Gamma; \Gamma'_k \Rightarrow \Delta; \Delta'_k]$.

Then we complete the proof of Theorem 4.1 by induction on the structure of the cut formula C.

5. Soundness and completeness of $\mathbf{S_{BBI}}$

This section proves the soundness and completeness of the nested sequent calculus $\mathbf{S_{BBI}}$ with respect to the satisfaction relation in Section 3.2 (Theorems 5.1 and 5.2). It means that the syntactic provability of a sequent coincides with its semantic unsatisfiability, confirming the principle of proof by contradiction embodied in the definition of sequents.

Theorem 5.1 (Soundness)**.** *If $\Gamma \Rightarrow \Delta$, then $\not\models_{\mathcal{W}} \Gamma \Rightarrow \Delta$.*

Theorem 5.2 (Completeness)**.** *If $\not\models_{\mathcal{W}} \Gamma \Rightarrow \Delta$, then $\Gamma \Rightarrow \Delta$.*

The proof of soundness proceeds by induction on the structure of the proof of $\Gamma \Rightarrow \Delta$. The proof of completeness uses a translation of a sequent W into a formula $[\![W]\!]_{\mathrm{w}}$ in Boolean BI defined as follows:

$$\begin{array}{rcl} [\![\Gamma \Rightarrow \Delta]\!]_{\mathrm{w}} & = & [\![\Gamma]\!]_{\mathrm{g}} \wedge \neg [\![\Delta]\!]_{\mathrm{d}} \\ [\![\cdot]\!]_{\mathrm{g}} = \top & & [\![\cdot]\!]_{\mathrm{d}} = \bot \\ [\![\Gamma; S]\!]_{\mathrm{g}} = [\![\Gamma]\!]_{\mathrm{g}} \wedge [\![S]\!]_{\mathrm{s}} & & [\![\Delta; A]\!]_{\mathrm{d}} = [\![\Delta]\!]_{\mathrm{d}} \vee A \\ [\![A]\!]_{\mathrm{s}} = A & & \\ [\![\emptyset_{\mathsf{m}}]\!]_{\mathrm{s}} = \mathsf{I} & & \\ [\![W_1, W_2]\!]_{\mathrm{s}} = [\![W_1]\!]_{\mathrm{w}} \star [\![W_2]\!]_{\mathrm{w}} & & \\ [\![W_1\langle\!\langle W_2 \rangle\!\rangle]\!]_{\mathrm{s}} = \neg([\![W_1]\!]_{\mathrm{w}} \mathrel{-\!\!\star} \neg [\![W_2]\!]_{\mathrm{w}}) & & \end{array}$$

Recall that a sequent W is a description of a set of nodes with respect to a reference node. $[\![W]\!]_{\mathrm{w}}$ is essentially the same description as W, except that it specifies the relationship between nodes through the use of multiplicative connectives $\star$ and $-\!\!\star$ and negation $\neg$.[2] The translation is characterized by Propositions 5.3 and 5.4.

Proposition 5.3. $\not\models_{\mathcal{W}} W$ *if and only if* $\models \neg [\![W]\!]_{\mathrm{w}}$.

Proposition 5.4. *If* $\cdot \Rightarrow \neg [\![\Gamma \Rightarrow \Delta]\!]_{\mathrm{w}}$, *then* $\Gamma \Rightarrow \Delta$.

Propositions 5.3 and 5.4 allow us to complete the proof of completeness if we additionally show that $\models A$ implies $\cdot \Rightarrow A$ (for $A = \neg [\![\Gamma \Rightarrow \Delta]\!]_{\mathrm{w}}$). Since $\models A$ implies $\vdash A$ by Theorem 2.1, it suffices to prove the following lemma, whose proof exploits the cut elimination property of $\mathbf{S_{BBI}}$ (Theorem 4.1):

Lemma 5.5. *If* $\vdash A$, *then* $\cdot \Rightarrow A$.

The following corollary shows that $\mathbf{S_{BBI}}$ is sound and complete with respect to the Kripke semantics in Section 2:

Corollary 5.6. $\cdot \Rightarrow A$ *if and only if* $\models A$.

6. Display calculus for Boolean BI

This section reviews the display calculus $\mathbf{DL_{BBI}}$ for Boolean BI (without the cut rule) by Brotherston [10]. We establish the equivalence between $\mathbf{S_{BBI}}$ and $\mathbf{DL_{BBI}}$, and show that $\mathbf{S_{BBI}}$ is an optimization of $\mathbf{DL_{BBI}}$.

6.1 Definition and properties of $\mathbf{DL_{BBI}}$

The display calculus $\mathbf{DL_{BBI}}$ uses a judgment $X \vdash_{\mathcal{D}} Y$, called a *consecution*, in its inference rules; X is called an A-structure and Y a C-structure:

$$\begin{array}{rcll} A\text{-structure} & X & ::= & A \mid \emptyset_{\mathsf{a}} \mid \emptyset_{\mathsf{m}} \mid \sharp Y \mid X; X \mid X, X \\ C\text{-structure} & Y & ::= & A \mid \emptyset_{\mathsf{a}} \mid \sharp X \mid Y; Y \mid X \mathrel{-\!\!\circ} Y \end{array}$$

A-structures are essentially an extension of bunches in Boolean BI with negative structures $\sharp Y$. C-structures do not use the multiplicative unit $\emptyset_{\mathsf{m}}$ and the multiplicative structural connective ,, but introduce a negative structural connective $\sharp$ and a multiplicative structural connective $-\!\!\circ$ which is originally from the display calculus for linear logic [2].

The inference rules of $\mathbf{DL_{BBI}}$ are divided into three groups: structural rules, display rules, and logical rules. Structural rules deal with the structural properties of consecutions. Display rules introduce or eliminate $\sharp Y$, $\sharp X$, and $X \mathrel{-\!\!\circ} Y$ as necessary in order to "display" a target A-structure or C-structure as the sole element in the left or right side of a consecution. A logical rule focuses on a single formula that has already been "displayed" in the left or right side of a consecution by the display rules. We refer the reader to [10] for the inference rules of $\mathbf{DL_{BBI}}$.

Brotherston [10] presents the following results on $\mathbf{DL_{BBI}}$:[3]

[2] The definition shows that $W_1\langle\!\langle W_2 \rangle\!\rangle$ is a meta-level operator corresponding to the logical connective *septraction* in [8, 37].

[3] Similarly to Theorem 4.1, Theorem 6.1 states only the admissibility of the cut rule, but we refer to it as the cut elimination theorem.

Theorem 6.1 (Cut elimination)**.**
If $X \vdash_{\mathcal{D}} A$ and $A \vdash_{\mathcal{D}} Y$, then $X \vdash_{\mathcal{D}} Y$.

Theorem 6.2 (Soundness and completeness)**.**
$\emptyset_{\mathsf{a}} \vdash_{\mathcal{D}} A$ if and only if $\models A$.

6.2 Equivalence between $\mathbf{S_{BBI}}$ and $\mathbf{DL_{BBI}}$

Although the equivalence between $\mathbf{S_{BBI}}$ and $\mathbf{DL_{BBI}}$ is obvious from Corollary 5.6 and Theorem 6.2, it does not illuminate how sequents and consecutions are related. Here we present direct translations between the two calculi and study their similarities and differences.

Given a consecution $X \vdash_{\mathcal{D}} Y$, we translate A-structure X to a sequent $[\![X]\!]_{\mathcal{X}}$ and C-structure Y to another sequent $[\![Y]\!]_{\mathcal{Y}}$. Then we combine $[\![X]\!]_{\mathcal{X}}$ and $[\![Y]\!]_{\mathcal{Y}}$ to build a single sequent $[\![X \vdash_{\mathcal{D}} Y]\!]_{\mathcal{C}}$. Let us write $(\Gamma \Rightarrow \Delta) \uplus (\Gamma' \Rightarrow \Delta')$ for $\Gamma; \Gamma' \Rightarrow \Delta; \Delta'$. We define $[\![X \vdash_{\mathcal{D}} Y]\!]_{\mathcal{C}}$ as follows:

$$
\begin{array}{rcl}
[\![X \vdash_{\mathcal{D}} Y]\!]_{\mathcal{C}} & = & [\![X]\!]_{\mathcal{X}} \uplus [\![Y]\!]_{\mathcal{Y}} \\
[\![A]\!]_{\mathcal{X}} & = & A \Rightarrow \cdot \\
[\![\emptyset_{\mathsf{a}}]\!]_{\mathcal{X}} & = & \cdot \Rightarrow \cdot \\
[\![\emptyset_{\mathsf{m}}]\!]_{\mathcal{X}} & = & \emptyset_{\mathsf{m}} \Rightarrow \cdot \\
[\![\sharp Y]\!]_{\mathcal{X}} & = & [\![Y]\!]_{\mathcal{Y}} \\
[\![X_1; X_2]\!]_{\mathcal{X}} & = & [\![X_1]\!]_{\mathcal{X}} \uplus [\![X_2]\!]_{\mathcal{X}} \\
[\![X_1, X_2]\!]_{\mathcal{X}} & = & [\![X_1]\!]_{\mathcal{X}}, [\![X_2]\!]_{\mathcal{X}} \Rightarrow \cdot \\
[\![A]\!]_{\mathcal{Y}} & = & \cdot \Rightarrow A \\
[\![\emptyset_{\mathsf{a}}]\!]_{\mathcal{Y}} & = & \cdot \Rightarrow \cdot \\
[\![\sharp X]\!]_{\mathcal{Y}} & = & [\![X]\!]_{\mathcal{X}} \\
[\![Y_1; Y_2]\!]_{\mathcal{Y}} & = & [\![Y_1]\!]_{\mathcal{Y}} \uplus [\![Y_2]\!]_{\mathcal{Y}} \\
[\![X \multimap Y]\!]_{\mathcal{Y}} & = & [\![X]\!]_{\mathcal{X}} \langle\!\langle [\![Y]\!]_{\mathcal{Y}} \rangle\!\rangle \Rightarrow \cdot
\end{array}
$$

Like sequents in $\mathbf{S_{BBI}}$, both A-structure X and C-structure Y are essentially descriptions of a set of nodes, but formulas in X are regarded as true whereas formulas in Y as false in the reference node. We also observe that multiplicative structures X_1, X_2 and $X \multimap Y$ correspond to multiplicative pairs and adjoint pairs in $\mathbf{S_{BBI}}$.

Lemma 6.3 shows that $\mathbf{S_{BBI}}$ is as expressive as $\mathbf{DL_{BBI}}$:

Lemma 6.3.
If $X \vdash_{\mathcal{D}} Y$ holds in $\mathbf{DL_{BBI}}$, then $[\![X \vdash_{\mathcal{D}} Y]\!]_{\mathcal{C}}$ holds in $\mathbf{S_{BBI}}$.

Given a sequent $\Gamma \Rightarrow \Delta$, we translate Γ to an A-structure $[\![\Gamma]\!]_{\mathcal{G}}$ and Δ to a C-structure $[\![\Delta]\!]_{\mathcal{D}}$. Then we combine $[\![\Gamma]\!]_{\mathcal{G}}$ and $[\![\Delta]\!]_{\mathcal{D}}$ to another A-structure $[\![\Gamma \Rightarrow \Delta]\!]_{\mathcal{W}}$ defined as follows:

$$
[\![\Gamma \Rightarrow \Delta]\!]_{\mathcal{W}} \quad = \quad [\![\Gamma]\!]_{\mathcal{G}}; \sharp [\![\Delta]\!]_{\mathcal{D}}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
[\![\cdot]\!]_{\mathcal{G}} & = & \emptyset_{\mathsf{a}} & [\![\cdot]\!]_{\mathcal{D}} & = & \emptyset_{\mathsf{a}} \\
[\![\Gamma; S]\!]_{\mathcal{G}} & = & [\![\Gamma]\!]_{\mathcal{G}}; [\![S]\!]_{\mathcal{S}} & [\![\Delta; A]\!]_{\mathcal{D}} & = & [\![\Delta]\!]_{\mathcal{D}}; A \\
[\![A]\!]_{\mathcal{S}} & = & A & & & \\
[\![\emptyset_{\mathsf{m}}]\!]_{\mathcal{S}} & = & \emptyset_{\mathsf{m}} & & & \\
[\![W_1, W_2]\!]_{\mathcal{S}} & = & [\![W_1]\!]_{\mathcal{W}}, [\![W_2]\!]_{\mathcal{W}} & & & \\
[\![W_1 \langle\!\langle W_2 \rangle\!\rangle]\!]_{\mathcal{S}} & = & \sharp([\![W_1]\!]_{\mathcal{W}} \multimap \sharp [\![W_2]\!]_{\mathcal{W}}) & & &
\end{array}
$$

$[\![W]\!]_{\mathcal{W}}$ is defined in a similar way to $[\![W]\!]_{\mathrm{w}}$ given in Section 5. For example, $[\![W_1, W_2]\!]_{\mathcal{S}}$ coincides with $[\![W_1, W_2]\!]_{\mathrm{s}}$ if we write , as $\star$, and $[\![W_1 \langle\!\langle W_2 \rangle\!\rangle]\!]_{\mathcal{S}}$ coincides with $[\![W_1 \langle\!\langle W_2 \rangle\!\rangle]\!]_{\mathrm{s}}$ if we write $\sharp$ as $\neg$ and $\multimap$ as $-\!\star$. The comparison between $[\![W]\!]_{\mathcal{W}}$ and $[\![W]\!]_{\mathrm{w}}$ reveals a correspondence between structural connectives in $\mathbf{DL_{BBI}}$ and logical connectives in Boolean BI that complies with the translation from $\mathbf{DL_{BBI}}$ to Boolean BI given in [10].

Lemma 6.4 shows that $\mathbf{DL_{BBI}}$ is as expressive as $\mathbf{S_{BBI}}$. In conjunction with Lemma 6.3, it proves the equivalence between $\mathbf{S_{BBI}}$ and $\mathbf{DL_{BBI}}$.

Lemma 6.4. *If $\Gamma \Rightarrow \Delta$, then $[\![\Gamma \Rightarrow \Delta]\!]_{\mathcal{W}} \vdash_{\mathcal{D}} \emptyset_{\mathsf{a}}$.*

With Lemmas 6.3 and 6.4, we can now give another indirect proof of Theorem 4.1 by exploiting Theorem 6.1. We need an additional lemma relating the two translations:

Lemma 6.5.
$[\![[\![\Gamma]\!]_{\mathcal{G}}]\!]_{\mathcal{X}} = \Gamma \Rightarrow \cdot$ and $[\![[\![\Delta]\!]_{\mathcal{D}}]\!]_{\mathcal{Y}} = \cdot \Rightarrow \Delta$.

Proof of Theorem 4.1. Suppose $\Gamma \Rightarrow \Delta; C$ and $\Gamma'; C \Rightarrow \Delta'$.
$[\![\Gamma \Rightarrow \Delta; C]\!]_{\mathcal{W}} \vdash_{\mathcal{D}} \emptyset_{\mathsf{a}}$ and $[\![\Gamma'; C \Rightarrow \Delta']\!]_{\mathcal{W}} \vdash_{\mathcal{D}} \emptyset_{\mathsf{a}}$ by Lemma 6.4
$[\![\Gamma]\!]_{\mathcal{G}}; \sharp [\![\Delta]\!]_{\mathcal{D}} \vdash_{\mathcal{D}} C$ and $C \vdash_{\mathcal{D}} \sharp [\![\Gamma']\!]_{\mathcal{G}}; [\![\Delta']\!]_{\mathcal{D}}$ by the display rules and the rule $\emptyset_{\mathsf{a}} R_{\mathcal{D}}$
$[\![\Gamma]\!]_{\mathcal{G}}; \sharp [\![\Delta]\!]_{\mathcal{D}} \vdash_{\mathcal{D}} \sharp [\![\Gamma']\!]_{\mathcal{G}}; [\![\Delta']\!]_{\mathcal{D}}$ by Theorem 6.1
$[\![\Gamma; \Gamma']\!]_{\mathcal{G}} \vdash_{\mathcal{D}} [\![\Delta; \Delta']\!]_{\mathcal{D}}$ by the display rules
$[\![[\![\Gamma; \Gamma']\!]_{\mathcal{G}} \vdash_{\mathcal{D}} [\![\Delta; \Delta']\!]_{\mathcal{D}}]\!]_{\mathcal{C}}$ in $\mathbf{S_{BBI}}$ by Lemma 6.3
$\Gamma; \Gamma' \Rightarrow \Delta; \Delta'$ by Lemma 6.5 □

6.3 $\mathbf{S_{BBI}}$ as an optimization of $\mathbf{DL_{BBI}}$

The two translations in Section 6.2 suggest that sequents in $\mathbf{S_{BBI}}$ essentially represent a normal form of consecutions in $\mathbf{DL_{BBI}}$. We say that a consecution is of the normal form if it permits a negative structure $\sharp X$ only in the right side of $\multimap$ according to the revised definition of C-structures:

$$
C\text{-structure} \quad Y \quad ::= \quad A \mid \emptyset_{\mathsf{a}} \mid Y; Y \mid X \multimap \sharp X
$$

It turns out that every consecution $X \vdash_{\mathcal{D}} Y$ can be converted by the structural rules (for associativity and $\emptyset_{\mathsf{a}}$) and the display rules to its normal form $[\![[\![X \vdash_{\mathcal{D}} Y]\!]_{\mathcal{C}}]\!]_{\mathcal{W}} \vdash_{\mathcal{D}} \emptyset_{\mathsf{a}}$, whose formulas form the same syntactic structure as the sequent $[\![X \vdash_{\mathcal{D}} Y]\!]_{\mathcal{C}}$. Thus we may think of sequents as representing consecutions of the normal form and $\mathbf{S_{BBI}}$ as a sequent calculus that directly manipulates such consecutions.

Note that consecutions of the normal form in $\mathbf{DL_{BBI}}$ still require the negative structural connective $\sharp$ whereas $\mathbf{S_{BBI}}$ requires no such negative structural connective. Hence those display rules dealing with $\sharp$ have no counterparts in $\mathbf{S_{BBI}}$, which implies that proof searches in $\mathbf{S_{BBI}}$ are always simpler than in $\mathbf{DL_{BBI}}$ (except in trivial cases) because of the extra cost of applying such display rules in $\mathbf{DL_{BBI}}$.

Figure 6 shows an example of proving in $\mathbf{DL_{BBI}}$ the same formula as in Figure 5. The proof search proceeds in a similar manner: first creating $\emptyset_{\mathsf{m}}$, next applying a contraction rule to duplicate a C-structure, then consuming the C-structure, and finally applying the rule $Init_{\mathcal{D}}$. We number each proof step to mark the correspondence between proof steps in Figures 5 and 6. Note that the display rule $MD1a_{\mathcal{D}}$ expands to a pair of a traverse rule ($TC_{\mathcal{S}}$ or $TP_{\mathcal{S}}$) and the rule $EC_{\mathcal{S}}$ (at proof steps 3, 5, and 9). We observe that $\mathbf{DL_{BBI}}$ takes extra six proof steps all of which apply display rules (marked in rectangles). This example illustrates that $\mathbf{S_{BBI}}$ is a formal system which can be obtained from an optimization of $\mathbf{DL_{BBI}}$ that dispenses with those display rules dealing with the negative structural connective $\sharp$ and revises all the logical rules accordingly.

7. Nested sequent calculus $\mathbf{CS_{BBI}}$

While the presence of multiplicative connectives from intuitionistic linear logic may suggest the inverse method for implementing a theorem prover for Boolean BI, the contraction property alone makes the inverse method not so ideal as it seems, as already observed in previous work on intuitionistic BI by Donnelly *et al.* [16]. This is especially the case for $\mathbf{S_{BBI}}$, which, unlike sequent calculi for intuitionistic BI, needs to use a graph structure of sequents instead of a tree structure. For example, it is not clear how to generate a minimal graph structure that weakens to a given pair of graph structures (for those inference rules with two sequents in the premise). Thus we choose to use a backward search strategy in our theorem prover.

As the first step, we obtain the nested sequent calculus $\mathbf{CS_{BBI}}$ (**C**ontraction-free $\mathbf{S_{BBI}}$), shown in Figure 7, by embedding the weakening and contraction rules ($WL_{\mathcal{S}}$, $WR_{\mathcal{S}}$, $CL_{\mathcal{S}}$, and $CR_{\mathcal{S}}$) into

Structural rules:

$$\frac{\Gamma;(\Gamma';W_1,W_2\Rightarrow\Delta'),W_3;W_1\oplus S_1,(W_2\oplus S_2,W_3\oplus S_3\Rightarrow\cdot)\Rightarrow\Delta}{\Gamma;(\Gamma';W_1,W_2\Rightarrow\Delta'),W_3\Rightarrow\Delta}\ EA_{\mathcal{C}}\quad \text{where}\ \begin{cases} S_1 = W_2\langle\!\langle\Gamma';W_3\langle\!\langle\Gamma\Rightarrow\Delta\rangle\!\rangle\Rightarrow\Delta'\rangle\!\rangle \\ S_2 = W_1\langle\!\langle\Gamma';W_3\langle\!\langle\Gamma\Rightarrow\Delta\rangle\!\rangle\Rightarrow\Delta'\rangle\!\rangle \\ S_3 = (\Gamma';W_1,W_2\Rightarrow\Delta')\langle\!\langle\Gamma\Rightarrow\Delta\rangle\!\rangle \end{cases}$$

$$\frac{\Gamma;W_2,W_1\Rightarrow\Delta}{\Gamma;W_1,W_2\Rightarrow\Delta}\ EC_{\mathcal{C}}\qquad \frac{\Gamma;(\Gamma\Rightarrow\Delta),(\emptyset_{\mathsf{m}}\Rightarrow\cdot)\Rightarrow\Delta}{\Gamma\Rightarrow\Delta}\ \emptyset_{\mathsf{m}}U_{\mathcal{C}}$$

$$\frac{\Gamma;(\Gamma_1\Rightarrow\Delta_1),(\Gamma_2;\emptyset_{\mathsf{m}}\Rightarrow\Delta_2);\Gamma_1;S\Rightarrow\Delta;\Delta_1}{\Gamma;(\Gamma_1\Rightarrow\Delta_1),(\Gamma_2;\emptyset_{\mathsf{m}}\Rightarrow\Delta_2)\Rightarrow\Delta}\ \emptyset_{\mathsf{m}}D_{\mathcal{C}}\quad \text{where } S=(\Gamma_2;\emptyset_{\mathsf{m}}\Rightarrow\Delta_2)\langle\!\langle\Gamma\Rightarrow\Delta\rangle\!\rangle$$

Traverse rules:

$$\frac{\Gamma_{\mathrm{c}1};(\Gamma_{\mathrm{c}2}\Rightarrow\Delta_{\mathrm{c}2})\langle\!\langle\Gamma\Rightarrow\Delta\rangle\!\rangle\Rightarrow\Delta_{\mathrm{c}1}}{\Gamma;(\Gamma_{\mathrm{c}1}\Rightarrow\Delta_{\mathrm{c}1}),(\Gamma_{\mathrm{c}2}\Rightarrow\Delta_{\mathrm{c}2})\Rightarrow\Delta}\ TC_{\mathcal{C}}\qquad \frac{\Gamma_{\mathrm{p}};(\Gamma\Rightarrow\Delta),(\Gamma_{\mathrm{s}}\Rightarrow\Delta_{\mathrm{s}})\Rightarrow\Delta_{\mathrm{p}}}{\Gamma;(\Gamma_{\mathrm{s}}\Rightarrow\Delta_{\mathrm{s}})\langle\!\langle\Gamma_{\mathrm{p}}\Rightarrow\Delta_{\mathrm{p}}\rangle\!\rangle\Rightarrow\Delta}\ TP_{\mathcal{C}}\quad \text{(p for parent, c for child, s for sibling)}$$

Logical rules:

$$\frac{}{\Gamma;A\Rightarrow\Delta;A}\ Init_{\mathcal{C}}\qquad \frac{}{\Gamma;\bot\Rightarrow\Delta}\ \bot L_{\mathcal{C}}\qquad \frac{\Gamma\Rightarrow\Delta;A}{\Gamma;\neg A\Rightarrow\Delta}\ \neg L_{\mathcal{C}}\qquad \frac{\Gamma;A\Rightarrow\Delta}{\Gamma\Rightarrow\Delta;\neg A}\ \neg R_{\mathcal{C}}$$

$$\frac{\Gamma;A\Rightarrow\Delta\quad\Gamma;B\Rightarrow\Delta}{\Gamma;A\vee B\Rightarrow\Delta}\ \vee L_{\mathcal{C}}\qquad \frac{\Gamma\Rightarrow\Delta;A;B}{\Gamma\Rightarrow\Delta;A\vee B}\ \vee R_{\mathcal{C}}\qquad \frac{\Gamma;\emptyset_{\mathsf{m}}\Rightarrow\Delta}{\Gamma;\mathsf{I}\Rightarrow\Delta}\ \mathsf{I}L_{\mathcal{C}}\qquad \frac{}{\Gamma;\emptyset_{\mathsf{m}}\Rightarrow\Delta;\mathsf{I}}\ \mathsf{I}R_{\mathcal{C}}$$

$$\frac{\Gamma;(A\Rightarrow\cdot),(B\Rightarrow\cdot)\Rightarrow\Delta}{\Gamma;A\star B\Rightarrow\Delta}\ \star L_{\mathcal{C}}\qquad \frac{\Gamma;(\Gamma_1\Rightarrow\Delta_1;A),(\Gamma_2\Rightarrow\Delta_2)\Rightarrow\Delta;A\star B\quad\Gamma;(\Gamma_1\Rightarrow\Delta_1),(\Gamma_2\Rightarrow\Delta_2;B)\Rightarrow\Delta;A\star B}{\Gamma;(\Gamma_1\Rightarrow\Delta_1),(\Gamma_2\Rightarrow\Delta_2)\Rightarrow\Delta;A\star B}\ \star R_{\mathcal{C}}$$

$$\frac{\Gamma;(\Gamma_1\Rightarrow\Delta_1;A)\langle\!\langle\Gamma_2\Rightarrow\Delta_2\rangle\!\rangle;A\mathrel{-\!\!\star}B\Rightarrow\Delta\quad\Gamma;(\Gamma_1\Rightarrow\Delta_1)\langle\!\langle\Gamma_2;B\Rightarrow\Delta_2\rangle\!\rangle;A\mathrel{-\!\!\star}B\Rightarrow\Delta}{\Gamma;(\Gamma_1\Rightarrow\Delta_1)\langle\!\langle\Gamma_2\Rightarrow\Delta_2\rangle\!\rangle;A\mathrel{-\!\!\star}B\Rightarrow\Delta}\ \mathrel{-\!\!\star}L_{\mathcal{C}}\qquad \frac{\Gamma;(A\Rightarrow\cdot)\langle\!\langle\cdot\Rightarrow B\rangle\!\rangle\Rightarrow\Delta}{\Gamma\Rightarrow\Delta;A\mathrel{-\!\!\star}B}\ \mathrel{-\!\!\star}R_{\mathcal{C}}$$

Figure 7. Nested sequent calculus $\mathbf{CS_{BBI}}$. We define $(\Gamma\Rightarrow\Delta)\oplus S$ as $\Gamma;S\Rightarrow\Delta$ in the rule $EA_{\mathcal{C}}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B\vdash_{\mathcal{D}}B}\ Init_{\mathcal{D}}{}^{15}}{B;\emptyset_{\mathsf{m}}\vdash_{\mathcal{D}}B}\ WL_{\mathcal{D}}{}^{14}}{\emptyset_{\mathsf{m}}\vdash_{\mathcal{D}}\sharp B;B}\ AD1a_{\mathcal{D}}}{\emptyset_{\mathsf{m}};\sharp B\vdash_{\mathcal{D}}\sharp B}\ AD2a_{\mathcal{D}}}{\emptyset_{\mathsf{m}};\sharp B\vdash_{\mathcal{D}}\neg B}\ \neg R_{\mathcal{D}}{}^{13}}{A,(\emptyset_{\mathsf{m}};\sharp B)\vdash_{\mathcal{D}}A\star\neg B}\ \star R_{\mathcal{D}}{}^{11}}{(A,(\emptyset_{\mathsf{m}};\sharp B));\sharp(A\star B)\vdash_{\mathcal{D}}A\star\neg B}\ WL_{\mathcal{D}}{}^{10}}{A,(\emptyset_{\mathsf{m}};\sharp B)\vdash_{\mathcal{D}}A\star B;A\star\neg B}\ AD2b_{\mathcal{D}}}{\emptyset_{\mathsf{m}};\sharp B\vdash_{\mathcal{D}}A\mathrel{-\!\!\star}(A\star B;A\star\neg B)}\ MD1a_{\mathcal{D}}{}^{9}}{\emptyset_{\mathsf{m}}\vdash_{\mathcal{D}}B;A\mathrel{-\!\!\star}(A\star B;A\star\neg B)}\ AD2b_{\mathcal{D}}}{\emptyset_{\mathsf{m}};\sharp(A\mathrel{-\!\!\star}(A\star B;A\star\neg B))\vdash_{\mathcal{D}}B}\ AD2a_{\mathcal{D}}}{A,(\emptyset_{\mathsf{m}};\sharp(A\mathrel{-\!\!\star}(A\star B;A\star\neg B)))\vdash_{\mathcal{D}}A\star B}\ \star R_{\mathcal{D}}{}^{7}}{A,(\emptyset_{\mathsf{m}};\sharp(A\mathrel{-\!\!\star}(A\star B;A\star\neg B)))\vdash_{\mathcal{D}}A\star B;A\star\neg B}\ WR_{\mathcal{D}}{}^{6}}{\emptyset_{\mathsf{m}};\sharp(A\mathrel{-\!\!\star}(A\star B;A\star\neg B))\vdash_{\mathcal{D}}A\mathrel{-\!\!\star}(A\star B;A\star\neg B)}\ MD1a_{\mathcal{D}}{}^{5}}{\emptyset_{\mathsf{m}}\vdash_{\mathcal{D}}(A\mathrel{-\!\!\star}(A\star B;A\star\neg B));(A\mathrel{-\!\!\star}(A\star B;A\star\neg B))}\ AD2a_{\mathcal{D}}}{\emptyset_{\mathsf{m}}\vdash_{\mathcal{D}}A\mathrel{-\!\!\star}(A\star B;A\star\neg B)}\ CR_{\mathcal{D}}{}^{4}}{A,\emptyset_{\mathsf{m}}\vdash_{\mathcal{D}}A\star B;A\star\neg B}\ MD1a_{\mathcal{D}}{}^{3}}{A\vdash_{\mathcal{D}}A\star B;A\star\neg B}\ \emptyset_{\mathsf{m}}U_{\mathcal{D}}{}^{2}}{A\vdash_{\mathcal{D}}(A\star B)\vee(A\star\neg B)}\ \vee R_{\mathcal{D}}{}^{1}$$

Figure 6. A proof of $A\vdash_{\mathcal{D}}(A\star B)\vee(A\star\neg B)$ in $\mathbf{DL_{BBI}}$ (adapted from Brotherston [9])

all the other rules of $\mathbf{S_{BBI}}$. Similarly to $\mathbf{S_{BBI}}$, the inference rules are divided into structural rules, traverse rules, and logical rules. Since contraction is built into all the inference rules in $\mathbf{CS_{BBI}}$, the premise of every structural rule subsumes the graph structure of nodes represented by the conclusion. Except the rule $EC_{\mathcal{C}}$ which rewrites a node state according to commutativity of sequents, every structural rule has a premise that strictly extends its conclusion with new relations between nodes. Below we further describe the two structural rules $EA_{\mathcal{C}}$ and $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$, in which the premise restructures nodes by changing parent-child and sibling relations but is also able to recover the original structure of nodes expressed in the conclusion.

In the rule $EA_{\mathcal{C}}$, each sequent $W_i\oplus S_i$ $(i=1,2,3)$ in the premise represents the same graph structure as the conclusion, except that its reference node is now described by W_i. Extending W_i with S_i in this way is a part of building contraction into the rule $EA_{\mathcal{C}}$ and is thus necessary for the completeness of $\mathbf{CS_{BBI}}$ with respect to Boolean BI. As an example, consider the proof of a sequent

$$(W_1,W_2\Rightarrow B),W_3\Rightarrow\cdot$$

where $W_1=A\Rightarrow\cdot$ and $W_2=C\mathrel{-\!\!\star}\neg(\neg(A\mathrel{-\!\!\star}B)\star C)\Rightarrow\cdot$ and $W_3=C\Rightarrow\cdot$. We apply the rule $EA_{\mathcal{C}}$ to generate another sequent

$$(W_1,W_2\Rightarrow B),W_3;W_1\oplus S_1,(W_2\oplus S_2,W_3\oplus S_3\Rightarrow\cdot)\Rightarrow\cdot$$

with $S_1=W_2\langle\!\langle W_3\langle\!\langle\cdot\Rightarrow\cdot\rangle\!\rangle\Rightarrow B\rangle\!\rangle$ and $S_2=W_1\langle\!\langle W_3\langle\!\langle\cdot\Rightarrow\cdot\rangle\!\rangle\Rightarrow B\rangle\!\rangle$ and $S_3=(W_1,W_2\Rightarrow B)\langle\!\langle\cdot\Rightarrow\cdot\rangle\!\rangle$. Eventually we reach the following sequent which is provable only because of the interaction between S_2 and $A\mathrel{-\!\!\star}B$ via the rule $\mathrel{-\!\!\star}L_{\mathcal{C}}$:

$$\begin{array}{l} C\mathrel{-\!\!\star}\neg(\neg(A\mathrel{-\!\!\star}B)\star C);\\ \quad(W_3\oplus S_3)\langle\!\langle(W_1\oplus S_1)\langle\!\langle S_1\Rightarrow\cdot\rangle\!\rangle\Rightarrow\neg(A\mathrel{-\!\!\star}B)\star C\rangle\!\rangle;\\ \qquad\boxed{S_2;A\mathrel{-\!\!\star}B}\Rightarrow\cdot \end{array}$$

Hence, if we omit S_i in the premise of the rule $EA_{\mathcal{C}}$, we lose the completeness of $\mathbf{CS_{BBI}}$.

The rule $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$ creates a new child node with a special form of sequent $\emptyset_{\mathsf{m}}\Rightarrow\cdot$, which can be absorbed back into the parent node by the rule $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$. Intuitively $\emptyset_{\mathsf{m}}\Rightarrow\cdot$ describes an empty node whose sibling node can be identified with its parent node. Similarly to the rule $EA_{\mathcal{C}}$, the premise of the rule $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$ combines the conclusion with a new sequent $\Gamma_1;S\Rightarrow\Delta_1$, which represents the same graph structure as the conclusion but has a different reference node. Omitting S in the premise also costs the completeness of $\mathbf{CS_{BBI}}$. For example, if we omit S in the premise of the rule $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$, the sequent in Figure 8 is not provable because its proof depends on the interaction between $A\mathrel{-\!\!\star}B$ and S via the rule $\mathrel{-\!\!\star}L_{\mathcal{C}}$.

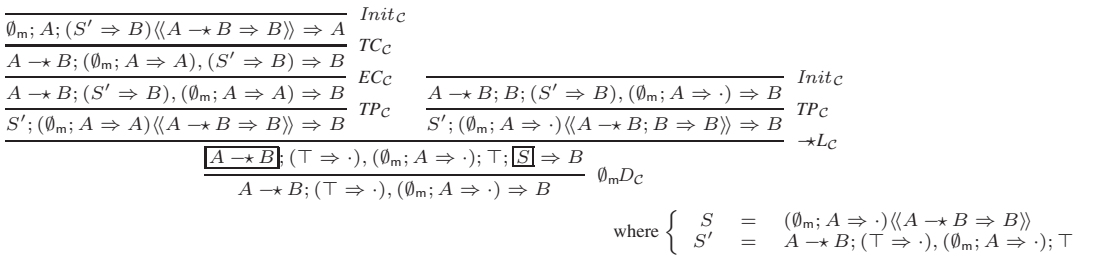


Figure 8. A proof of $A \mathrel{-\!\!\star} B; (\top \Rightarrow \cdot), (\emptyset_{\mathsf{m}}; A \Rightarrow \cdot) \Rightarrow B$ in $\mathbf{CS_{BBI}}$

Every inference rule in $\mathbf{CS_{BBI}}$ is invertible, *i.e.*, the premise implies the conclusion and vice versa. We can formally prove that both weakening and contraction are admissible in $\mathbf{CS_{BBI}}$. We can also prove the equivalence between $\mathbf{S_{BBI}}$ and $\mathbf{CS_{BBI}}$.

Theorem 7.1 (Weakening and contraction in $\mathbf{CS_{BBI}}$)**.**
If $\Gamma \Rightarrow \Delta$*, then* $\Gamma; S \Rightarrow \Delta$ *and* $\Gamma \Rightarrow \Delta; A$*.*
If $\Gamma; S; S \Rightarrow \Delta$*, then* $\Gamma; S \Rightarrow \Delta$*.*
If $\Gamma \Rightarrow \Delta; A; A$*, then* $\Gamma \Rightarrow \Delta; A$*.*

Theorem 7.2 (Equivalence between $\mathbf{S_{BBI}}$ and $\mathbf{CS_{BBI}}$)**.**
$\Gamma \Rightarrow \Delta$ *in* $\mathbf{S_{BBI}}$ *if and only if* $\Gamma \Rightarrow \Delta$ *in* $\mathbf{CS_{BBI}}$*.*

Figure 9 shows an example of proving in $\mathbf{CS_{BBI}}$ the same sequent as in Figure 5. The proof tree is much smaller: the depth decreases from 16 to 8 and the number of applications of rules decreases from 18 to 12. Besides the amount of non-determinism in proof search is now minimal. In Figure 5, after applying the rule $\emptyset_{\mathsf{m}} U_{\mathcal{S}}$ (when read from the conclusion to the premise), we have to decide whether or not to duplicate S by applying the contraction rule $CL_{\mathcal{S}}$, and if we skip the rule $CL_{\mathcal{S}}$, proof search fails. In Figure 9, this form of non-determinism does not arise because the contraction rule is embedded into the rule $\emptyset_{\mathsf{m}} U_{\mathcal{C}}$. As a result, except for applying the rule $\emptyset_{\mathsf{m}} U_{\mathcal{C}}$ indefinitely, the only source of non-determinism concerns which of $A \star B$ and $A \star \neg B$ should be considered first by the rule $\star R_{\mathcal{C}}$, which is irrelevant for the purpose of this proof anyway.

Since their role is to change only the reference node without changing the graph structure of nodes, the traverse rule $TC_{\mathcal{C}}$ and the structural rule $EC_{\mathcal{C}}$ in Figure 9 do not increase the complexity of proof search. For example, once we decide to focus on $\neg B$ in $A; (A \Rightarrow \Delta), (\emptyset_{\mathsf{m}} \Rightarrow B; \neg B) \Rightarrow \Delta$, we obtain a unique sequence of rules, namely $EC_{\mathcal{C}}$ followed by $TC_{\mathcal{C}}$, for exposing $\emptyset_{\mathsf{m}} \Rightarrow B; \neg B$ in the reference node. Thus the cost of proof search is incurred mainly by various decisions on applying the structural and logical rules and not by applications of the traverse rules. Section 8.5 explains how to eliminate the traverse rules altogether in proof search.

8. Backward proof search in $\mathbf{CS_{BBI}}$

This section explains the design of our theorem prover which uses a backward search strategy built on top of $\mathbf{CS_{BBI}}$. Because of the undecidability of Boolean BI [11, 29], our theorem prover implements a semi-decision algorithm. For our purpose, a semi-decision algorithm is still useful because in program verification, we usually attempt to prove formulas that are believed to be true.

Our theorem prover includes a certifier which converts every proof in $\mathbf{CS_{BBI}}$ into an equivalent proof in $\mathbf{S_{BBI}}$ according to our proof of Theorem 7.2. In addition to automated proof search, it also supports an interactive mode, in which the user can issue various tactics to manually change the structure of nodes. Both extensions are a preliminary step toward developing a program verification system based on full separation logic. For space reasons, we do not describe these extensions.

```
Algorithm ProveBBI(W, d)
  Input: goal sequent W, search depth d
  Output: true or fail
 1:  for each node w,
 2:    promote w as the reference node
 3:    if Init_C, ⊥L_C, or lR_C is applicable, return true
 4:    if ¬L_C, ¬R_C, ∨R_C, lL_C, ⋆L_C, or -⋆R_C is applicable,
 5:      W' ← result of applying the rule
 6:      return ProveBBI(W', d)
 7:    if ∨L_C is applicable,
 8:      W_1 and W_2 ← result of applying the rule
 9:      return ProveBBI(W_1, d) && ProveBBI(W_2, d)
10:    if ⋆R_C or -⋆L_C is applicable to a fresh node state S,
11:      W_1 and W_2 ← result of applying the rule to S
12:      return ProveBBI(W_1, d) && ProveBBI(W_2, d)
13:  if d = 0, return fail
14:  for each node state S in node w
                    to which EA_C or ∅_m D_C is applicable,
15:    promote w as the reference node
16:    W' ← result of applying the rule to S
17:    if ProveBBI(W', d − 1) = true, return true
18:  for each node w,
19:    promote w as the reference node
20:    W' ← result of applying ∅_m U_C
21:    if ProveBBI(W', d − 1) = true, return true
22:  return fail
```

Figure 10. Pseudocode for the proof search algorithm ProveBBI

8.1 Basic structure of the proof search algorithm

Figure 10 shows the pseudocode for the proof search algorithm ProveBBI. Given a goal sequent W and a search depth d as input, it attempts to search for a proof tree for W with at most d applications of the structural rules (except the rule $EC_{\mathcal{C}}$) along any search path. First it examines every formula in a given sequent and applies the corresponding logical rule if possible (lines 1–12). After checking if $d = 0$ (line 13), it considers the structural rules $EA_{\mathcal{C}}$, $\emptyset_{\mathsf{m}} D_{\mathcal{C}}$, and $\emptyset_{\mathsf{m}} U_{\mathcal{C}}$ (lines 14–21). It returns either **true** or **fail**, depending on the result of the proof search. For the sake of simplicity, the algorithm ProveBBI assumes that the rule $EC_{\mathcal{C}}$ is embedded into all the other rules. For example, the rule $EA_{\mathcal{C}}$ in Figure 10 indeed refers to one of the four instances obtained by independently applying the rule $EC_{\mathcal{C}}$ to node states W_1, W_2 and $(\Gamma'; W_1, W_2 \Rightarrow \Delta'), W_3$.

The first phase of the algorithm ProveBBI (lines 1–12) exploits the fact that all the logical rules, including $\star R_{\mathcal{C}}$ and $\mathrel{-\!\!\star} L_{\mathcal{C}}$, are invertible. Hence it starts by applying the logical rules wherever possible until no more applications are left. For the rules $\star R_{\mathcal{C}}$ and $\mathrel{-\!\!\star} L_{\mathcal{C}}$, we make sure that they do not focus more than once on the

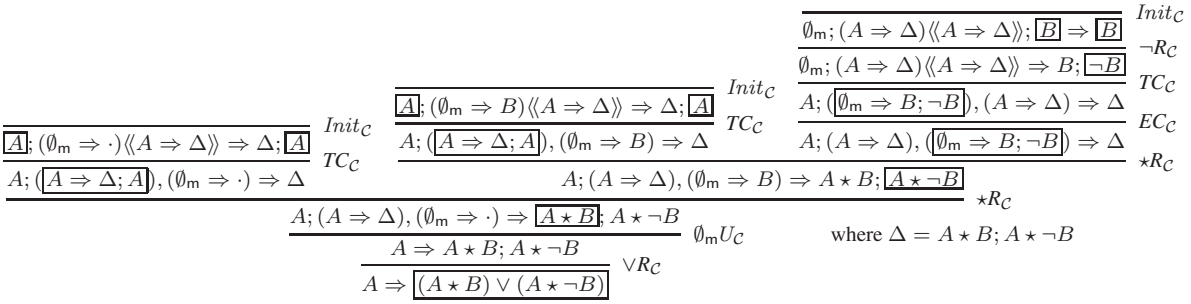


Figure 9. A proof of $A \Rightarrow (A \star B) \vee (A \star \neg B)$ in $\mathbf{CS_{BBI}}$. Each rectangle marks a principal formula or a sequent to be exposed in the reference node.

same pair of a principal formula and a node state (*e.g.*, a pair of $A \mathrel{-\!\!\star} B$ and $(\Gamma_1 \Rightarrow \Delta_1), (\Gamma_2 \Rightarrow \Delta_2)$ in the rule $\star R_{\mathcal{C}}$), since the principal formula survives in the premise.

After the first phase, the algorithm ProveBBI recursively invokes itself to apply the structural rules $EA_{\mathcal{C}}$, $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$, and $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$ in depth-first order (lines 14–21). For example, with $d = 2$, it considers all sequences of length 2 in the following order: $EA_{\mathcal{C}}$–$EA_{\mathcal{C}}$, $EA_{\mathcal{C}}$–$\emptyset_{\mathsf{m}}D_{\mathcal{C}}$, $EA_{\mathcal{C}}$–$\emptyset_{\mathsf{m}}U_{\mathcal{C}}$, $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$–$EA_{\mathcal{C}}$, $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$–$\emptyset_{\mathsf{m}}D_{\mathcal{C}}$, $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$–$\emptyset_{\mathsf{m}}U_{\mathcal{C}}$, $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$–$EA_{\mathcal{C}}$, $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$–$\emptyset_{\mathsf{m}}D_{\mathcal{C}}$, $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$–$\emptyset_{\mathsf{m}}U_{\mathcal{C}}$. We choose to consider the rules $EA_{\mathcal{C}}$ and $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$ before the rule $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$, which is the least restrictive rule in the sense that it can be applied to any sequent. Note that applying a structural rule does not create new formulas but only gives rise to new pairs of a formula ($A \star B$ or $A \mathrel{-\!\!\star} B$) and a node state to which the rules $\star R_{\mathcal{C}}$ and $\mathrel{-\!\!\star} L_{\mathcal{C}}$ can be applied. A recursive invocation of ProveBBI immediately focuses on these new pairs during its first phase (lines 10–12).

8.2 Explosion in the search space and the number of subgoals

While the algorithm ProveBBI eventually finds a proof tree for every provable sequent if given an unlimited search depth, a naive implementation suffers from two problems that are unique to Boolean BI. The first is an explosion in the search space in terms of the amount of conjunctive non-determinism among the logical rules $\star R_{\mathcal{C}}$ and $\mathrel{-\!\!\star} L_{\mathcal{C}}$ and the structural rules. That is, a typical proof search is quickly overwhelmed with too many choices for applying these rules, which are all invertible and thus can be applied aggressively. This problem is due to the structural rules $EA_{\mathcal{C}}$, $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$, and $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$, each application of which immediately doubles or quadruples the search space. The second problem is an explosion in the number of subgoals due to the logical rules $\star R_{\mathcal{C}}$ and $\mathrel{-\!\!\star} L_{\mathcal{C}}$, each application of which increments the number of subgoals. These rules can be applied to each formula $A \star B$ or $A \mathrel{-\!\!\star} B$ as many times as there are corresponding node states. For example, if Γ contains n multiplicative pairs, the rule $\star R_{\mathcal{C}}$ creates 2^n subgoals from $\Gamma \Rightarrow A \star B$ during the first phase of the algorithm ProveBBI.

The first problem is closely related to the contraction property built into the structural rules. As an example, consider the structural rule $EA_{\mathcal{C}}$ in Figure 7. The graph structure of the premise has four times more nodes than that of the conclusion because each sequent $W_i \oplus S_i$ ($i = 1, 2, 3$) represents the same graph structure as the conclusion, as shown in Figure 11. Consequently the premise provides four times more ways to apply the rules of $\mathbf{CS_{BBI}}$, thus quadrupling the search space. In a similar way, each application of the other structural rules $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$ and $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$ doubles the search space. We remark that the first problem is *not* the price to pay for building contraction into the structural rules, since the same problem of search space explosion remains even if we do not build contraction into the structural rules.

The second problem itself is orthogonal to the first problem, but its effect is heavily exacerbated by the first problem. As an example, consider again the structural rule $EA_{\mathcal{C}}$ in Figure 7 where we set $\Delta = A \star (B \star C)$. After an application of the rule $EA_{\mathcal{C}}$ to obtain the graph structure in Figure 11, two applications of the rule $\star R_{\mathcal{C}}$ ensue to copy A to W_1, B to W_2, and C to W_3. If these formulas happen to involve multiplicative connectives, we can again apply the rules $\star R_{\mathcal{C}}$ and $\mathrel{-\!\!\star} L_{\mathcal{C}}$ to propagate their component formulas, which, in turn, may trigger further applications of the rules $\star R_{\mathcal{C}}$ and $\mathrel{-\!\!\star} L_{\mathcal{C}}$, and so on.

To alleviate the first problem, we need to devise a scheme for prioritizing applications of the structural rules (Section 8.3). We can solve the second problem by borrowing an idea from the inverse method (Section 8.4). In addition, we can eliminate the traverse rules altogether (Section 8.5).

8.3 Prioritizing applications of the structural rules

As a solution to the first problem, we assign a priority, either high or low, to every sibling relation between nodes so as to prioritize all applications of the rules $EA_{\mathcal{C}}$ and $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$, and to every node itself so as to prioritize all applications of the rule $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$. When applying a structural rule, the algorithm ProveBBI first considers sibling relations and nodes with a high priority and then those with a low priority. Below we explain how to assign priorities to sibling relations and how to determine priorities for nodes.

For the rule $EA_{\mathcal{C}}$ as shown in Figure 7, we assign priorities to sibling relations in the premise according to Figure 11 with the following interpretation:

- For a sequent inside a rectangle $\boxed{W}$, every sibling relation in it is assigned the same priority as in the conclusion.
- For a sequent inside a dashed rectangle $\boxed{W}$, every sibling relation in it is assigned a low priority.
- A sibling relation depicted with solid lines $\wedge$ is assigned a high priority.
- A sibling relation depicted with dashed lines $\wedge$ is assigned a low priority.

The rationale for this assignment is that an application of the rule $EA_{\mathcal{C}}$ is primarily intended to generate sibling relations described by $W_1, (W_2, W_3 \Rightarrow \cdot)$, rather than S_1, S_2, and S_3 in $W_1 \oplus S_1, (W_2 \oplus S_2, W_3 \oplus S_3 \Rightarrow \cdot)$. When applying the rule $EA_{\mathcal{C}}$, the algorithm ProveBBI focuses first on those node states both of whose sibling relations have a high priority. In a similar way, we assign priorities to sibling relations in the premise of the rules $\emptyset_{\mathsf{m}}D_{\mathcal{C}}$ and $\emptyset_{\mathsf{m}}U_{\mathcal{C}}$ according to Figure 12.

We determine priorities of nodes by analyzing priorities of sibling relations. If a node is involved in a sibling relation with a high

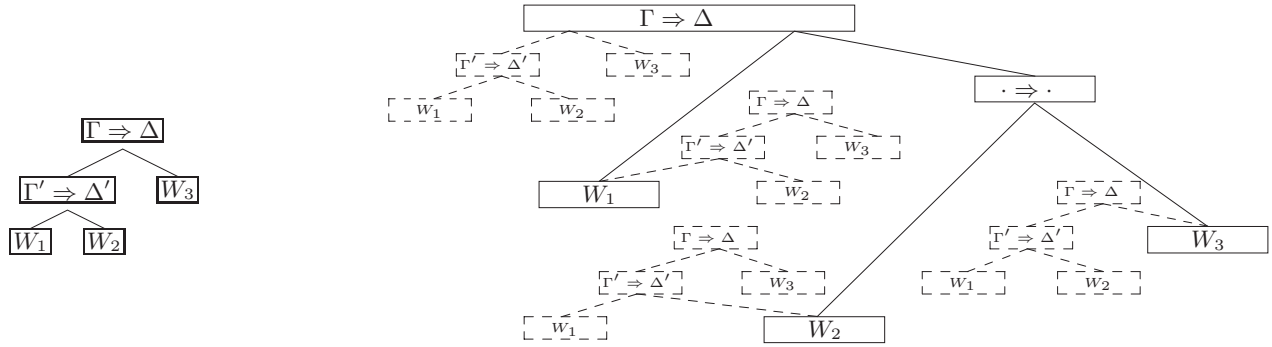
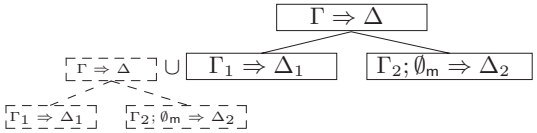


Figure 11. The graph structure of nodes before (conclusion) and after (premise) applying the structural rule $EA_{\mathcal{C}}$ in Figure 7

After applying the rule $\emptyset_{\mathsf{m}} D_{\mathcal{C}}$:



After applying the rule $\emptyset_{\mathsf{m}} U_{\mathcal{C}}$:

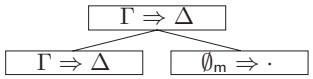


Figure 12. Assigning priorities to sibling relations after applying the rules $\emptyset_{\mathsf{m}} D_{\mathcal{C}}$ and $\emptyset_{\mathsf{m}} U_{\mathcal{C}}$. $(\Gamma \Rightarrow \Delta) \cup (\Gamma' \Rightarrow \Delta')$ is defined as $\Gamma; \Gamma' \Rightarrow \Delta; \Delta'$.

priority, it is more likely to be under active consideration by other structural rules than those nodes with no such involvement. Hence we assign a high priority to every node involved in at least one such sibling relation. For the logical rules $\star L_{\mathcal{C}}$ and $-\!\star R_{\mathcal{C}}$, we reuse the priority assigned to the reference node of the conclusion for two new nodes in the premise.

Now we redesign the algorithm ProveBBI as a two-stage algorithm. In the first stage, it applies the structural rules using only sibling relations and nodes with a high priority. Note that it still generates every node with a low priority, which is never used by the structural rules, but may be needed by the logical rules. For example, the two sequents discussed in Section 7 are provable in the first stage precisely because we also generate every node with a low priority. If the proof search fails, it enters the second stage and repeats the proof search without ignoring sibling relations and nodes with a low priority. The second stage is necessary for the completeness of proof search, since some formula requires us to apply the structural rules using those with a low priority as well. In fact, we can find even a formula whose proof tree applies the structural rules using *only* those with a low priority. Section 8.6 presents examples of such formulas.

8.4 Reusing the proof tree from the premise

We solve the problem of an explosion in the number of subgoals with a simple technique of reusing the proof tree from the premise. Suppose that we apply the rule $\star R_{\mathcal{C}}$ or $-\!\star L_{\mathcal{C}}$ to produce two subgoals in the premise, without knowing whether this application is necessary or not. If this application is unnecessary, however, every proof tree for the first premise must be a proof tree for the conclusion as well. Hence, upon finding a proof tree for the first subgoal, we "replay" it against the conclusion, and attempt to prove the second subgoal only in the case of a failure. In this way, we can aggressively apply the rules $\star R_{\mathcal{C}}$ and $-\!\star L_{\mathcal{C}}$ without worrying about an explosion in the number of subgoals. In essence, we partially simulate the inverse method with a moderate overhead of revisiting proof trees (but without entirely reformulating the nested sequent calculus $\mathbf{CS_{BBI}}$).

8.5 Eliminating the traverse rules

The algorithm ProveBBI invokes the traverse rules to change the reference node (lines 2, 15, 19 in Figure 10), but we can eliminate the traverse rules altogether with a slight change in the representation of sequents. The basic observation is that the traverse rules change only the reference node without altering the graph structure of nodes. Hence, by rewriting every rule in such a way that it directly focuses on any formula or node without requiring a reference node, we can discard the traverse rules.

To this end, we introduce a labelled sequent which assigns a unique label w to every node and annotates all formulas and $\emptyset_{\mathsf{m}}$'s in it with w:

$$\begin{array}{rrcl} \text{labelled sequent} & L & = & \Xi \vdash \Sigma \Rightarrow \Pi \\ \text{graph structure} & \Xi & ::= & \cdot \mid \Xi, w \sim w_1 \cdot w_2 \\ \text{labelled truth context} & \Sigma & ::= & \cdot \mid \Sigma, A@w \mid \Sigma, \emptyset_{\mathsf{m}}@w \\ \text{labelled falsehood context} & \Pi & ::= & \cdot \mid \Pi, A@w \end{array}$$

$w \sim w_1 \cdot w_2$ in the graph structure specifies that w is a parent node of w_1 and w_2. Then we can convert every sequent to a *unique* labelled sequent modulo renaming labels, since a sequent determines a unique graph structure of nodes where each node contains a unique set of true formulas, $\emptyset_{\mathsf{m}}$'s, and false formulas. Let us write $[\![W]\!]$ for the unique labelled sequent converted from sequent W. For a rule deducing W from W' (and W'') in $\mathbf{CS_{BBI}}$, we derive a new rule that deduces $[\![W]\!]$ from $[\![W']\!]$ (and $[\![W'']\!]$) in a single step; for an axiom deducing W in $\mathbf{CS_{BBI}}$, we derive a new axiom deducing $[\![W]\!]$. We refer to the resultant system as the labelled $\mathbf{CS_{BBI}}$:

$$\frac{W'}{W} R \quad \text{in } \mathbf{CS_{BBI}} \quad \longrightarrow \quad \frac{[\![W']\!]}{[\![W]\!]} R_{\mathcal{L}} \quad \text{in the labelled } \mathbf{CS_{BBI}}$$

The labelled $\mathbf{CS_{BBI}}$ has no traverse rules because the premise and conclusion of a traverse rule in $\mathbf{CS_{BBI}}$ represent the same graph structure of nodes. Still it is equivalent to $\mathbf{CS_{BBI}}$ because the definition of labelled sequents embeds the traverse rules into all the inference rules:

Proposition 8.1. *W is provable in $\mathbf{CS_{BBI}}$ if and only if $[\![W]\!]$ is provable in the labelled $\mathbf{CS_{BBI}}$.*

Figure 13 shows an example of proving in the labelled $\mathbf{CS_{BBI}}$ the same formula as in Figure 9. The depth decreases from 8 to

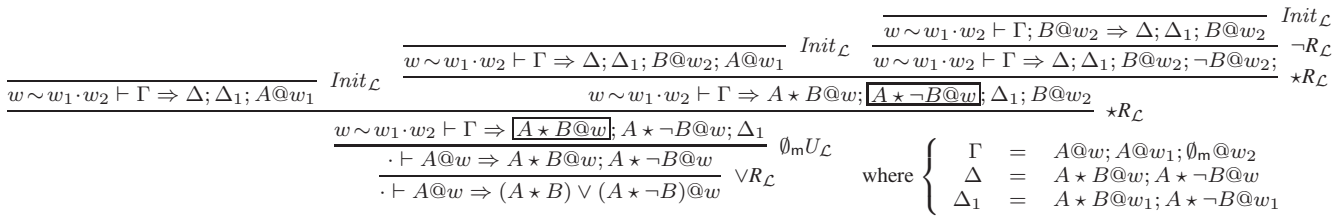


Figure 13. A proof of $\cdot \vdash A@w \Rightarrow (A \star B) \vee (A \star \neg B)@w$ in the labelled $\mathbf{CS_{BBI}}$. The rule $\star R_{\mathcal{L}}$ focuses on the formula inside the rectangle.

6 and the number of applications of rules decreases from 12 to 8. The rule $Init_{\mathcal{L}}$ immediately completes the proof when it detects the same labelled formula in both contexts of a given labelled sequent. The rule $\neg R_{\mathcal{L}}$ also directly focuses on $\neg B@w_2$, regardless of the presence of $w \sim w_1 \cdot w_2$ in the graph structure. In this way, the labelled $\mathbf{CS_{BBI}}$ dispenses with the traverse rules, yielding a smaller proof tree than $\mathbf{CS_{BBI}}$.

8.6 Experimental results

We compare a naive implementation of the algorithm ProveBBI with an optimized implementation that incorporates those ideas described in Sections 8.3 and 8.4. Both implementations internally use the labelled $\mathbf{CS_{BBI}}$ to eliminate the traverse rules, as explained in Section 8.5. Our implementations are written in Objective CAML and run on Ubuntu Linux 11.10 with Intel Core i7-960 3.2GHz and 6 gigabytes of main memory.

Figure 14 shows results of running both implementations (*naive* and *optimized*) on 14 representative formulas. For a given formula A, we use sequent $\cdot \Rightarrow A$ and search depth d as input to the algorithm ProveBBI. Except for experiment (c), we set d to the minimum search depth for finding a proof tree. The result is either the return value of ProveBBI (**true** and **fail**) or **error** if the proof search does not terminate within 10 minutes. In measuring the cost in terms of the number of applications of the rules, we exclude the rule $EC_{\mathcal{C}}$ which is already embedded into all the other rules. The elapsed time is in seconds.

Experiment (a) tests nine formulas (all involving multiplicative connectives) of increasing complexity. The two formulas marked ♯ require only those applications of the rule $EA_{\mathcal{C}}$ in which sibling relations with a low priority are visited; hence the proof search finishes in the second stage of the algorithm ProveBBI. Experiment (b) is designed to measure the effectiveness of the two optimizations specifically against the rule $EA_{\mathcal{C}}$. Experiment (c) tests the effect of increasing d for a common formula which can be proven with two applications of the rule $EA_{\mathcal{C}}$ followed by an application of the rule $\emptyset_{\mathsf{m}} D_{\mathcal{C}}$.

We observe that the cost of proof search is mainly driven by search depth d, *i.e.*, the number of applications of the structural rules required to complete proof search. We also observe that the optimized implementation is much less susceptible to the exponential growth of the search space than the naive implementation, thereby demonstrating that the two optimizations in Sections 8.3 and 8.4 are indeed highly effective. In experiment (c), a search depth of 3 eventually produces a proof tree, but only after a number of wrong applications of the rule $EA_{\mathcal{C}}$. An increase of d to 4, however, immediately incurs a wrong application of the rule $EA_{\mathcal{C}}$ which happens to lead to the correct two applications of the rule $EA_{\mathcal{C}}$, which is why it produces a proof tree at a much lower cost (from 5942 to 181). A further increase of d to 5 does not significantly increase the cost, but the elapsed time becomes much longer because of the extra overhead of manipulating much larger sequents.

Overall we find that the optimized implementation is reasonably fast in proving typical formulas of Boolean BI.

9. Related work

9.1 Proof search in the logic of BI and separation logic

Previous work on proof search in the logic of BI mainly focuses on intuitionistic BI, which is another member in the family that inherits its multiplicative connectives from intuitionistic linear logic (like Boolean BI), but additive connectives from intuitionistic propositional logic. Galmiche and Méry [18, 19] present a labelled tableau calculus for a propositional fragment without $\bot$ and develop a theorem prover, called BILL, on top of it. Their later paper [21] extends the calculus for full intuitionistic BI. Donnelly *et al.* [16] investigate the inverse method for a propositional fragment without units ($\top$, $\bot$, and I) and develop a forward theorem prover.

For Boolean BI, no theorem prover has been developed yet because of the lack of a proof theory suitable for proof search. Larchey-Wendling and Gamliche [28] formulate a labelled tableau calculus by extending the labelled tableau calculus for intuitionistic BI in [21], but only in order to investigate the relation between intuitionistic BI and Boolean BI. Brotherston [10] shows that a modular combination of display calculi for classical logic and intuitionistic linear logic gives rise to a display calculus $\mathbf{DL_{BBI}}$ for Boolean BI, the first cut-free syntactic formulation of Boolean BI, and proves the cut elimination property by observing that its rules obey all the syntactic constraints given in [1]. Developing a practical proof search strategy on top of it, however, is far from easy because of the complexity due to its display rules and the difficulty in restricting applications of the contraction rules [9].

Galmiche and Méry [20] present a labelled tableau calculus for separation logic. It lies somewhere between syntactic (tableau) and semantic (labelled) formulations because labels correspond to heaps in separation logic. Their calculus, albeit sound and complete, does not directly translate to a proof search strategy in its current form. It is easy to build a tableau for a given formula according to the calculus, but one needs to check if all branches in the tableau are logically or structurally inconsistent. This requires two semantic functions (a measure and an interpretation) for each branch, and the calculus does not specify how to obtain such semantic functions. For a similar reason, the labelled tableau calculus for Boolean BI in [28] does not directly translate to a proof search strategy. For theorem provers for the decidable fragment of separation logic by Berdine *et al*. [3] (without separating implication), see, for example, [4, 14, 32].

9.2 Nested sequent calculi

A nested sequent calculus is one whose sequent may contain smaller sequents. It has been used as a proof-theoretic formulation of some modal and tense logics [12, 27] for which no sequent calculus of the standard form exists. $\mathbf{S_{BBI}}$ is also a nested sequent calculus because a sequent may contain smaller sequents.

	formula	d	*naive*		*optimized*		
			result	cost	result	cost	time
(a)	$(A \mathrel{-\!\!\star} B) \wedge (\top \star (\mathsf{I} \wedge A)) \to B$	1	**true**	9	**true**	9	0.001
	$(\mathsf{I} \mathrel{-\!\!\star} \neg(\neg A \star \mathsf{I})) \to A$	1	**true**	9	**true**	9	0.001
	$\neg((A \mathrel{-\!\!\star} \neg(A \star B)) \wedge ((\neg A \mathrel{-\!\!\star} \neg B) \wedge B))$	1	**true**	26	**true**	19	0.001
	$\mathsf{I} \to (A \mathrel{-\!\!\star} B \mathrel{-\!\!\star} C) \mathrel{-\!\!\star} A \star B \mathrel{-\!\!\star} C$	2	**true**	700	**true**	76	0.021
	$\mathsf{I} \to A \star (B \star C) \mathrel{-\!\!\star} A \star B \star C$	2	**true**	2606	**true**	263	0.051
	$\mathsf{I} \to A \star ((B_1 \mathrel{-\!\!\star} B_2) \star C) \mathrel{-\!\!\star} A \star (B_1 \mathrel{-\!\!\star} B_2) \star C$	2	**true**	12317	**true**	336	0.102
♯	$\neg((A \mathrel{-\!\!\star} \neg(\neg(D \mathrel{-\!\!\star} \neg(A \star (C \star B))) \star A)) \wedge C \star (D \wedge (A \star B)))$	2	**true**	121000	**true**	380	0.053
	$\neg((C_1 \star (C_2 \star C_3)) \wedge$ $((A \mathrel{-\!\!\star} \neg(\neg(B \mathrel{-\!\!\star} \neg(C_2 \star (C_3 \star C_1))) \star A)) \star (B \wedge (A \star \top))))$	2	**true**	1614053	**true**	43	0.023
♯	$\neg((A \mathrel{-\!\!\star} \neg(\neg(D \mathrel{-\!\!\star} \neg(C \star B_2 \star (B_1 \star A))) \star A)) \wedge C$ $\star(D \wedge (A \star (B_1 \star B_2))))$	3	**error**	-	**true**	74856	128.4
(b)	$A \star (B \star (C \star D)) \to D \star (C \star (B \star A))$	2	**true**	2618	**true**	56	0.012
	$A \star (B \star (C \star D)) \to D \star (B \star (C \star A))$	3	**true**	15686	**true**	34	0.041
	$A \star (B \star (C \star (D \star E))) \to E \star (D \star (A \star (B \star C)))$	3	**error**	-	**true**	2320	2.2
	$A \star (B \star (C \star (D \star E))) \to E \star (B \star (A \star (C \star D)))$	4	**error**	-	**true**	2921	28.2
(c)	$\mathsf{I} \to A \star ((B_1 \mathrel{-\!\!\star} B_2) \star (C \star D)) \mathrel{-\!\!\star} A \star D \star (C \star (B_1 \mathrel{-\!\!\star} B_2))$	2	**fail**	1795	**fail**	1837	0.438
		3	**error**	-	**true**	5942	8.2
		4	**error**	-	**true**	181	7.0
		5	**error**	-	**true**	182	127.0
		6	**error**	-	**error**	-	-

Figure 14. Results of running two implementations (*naive* and *optimized*) of the algorithm ProveBBI. The elapsed time is in seconds.

A nested sequent calculus is often obtained as an optimization of an equivalent display calculus that is not simplified to a sequent calculus of the standard form. Gore *et al.* [22–24] propose such nested sequent calculi for bi-intuitionistic logic and classical tense logic. In particular, their nested sequent calculus $\mathbf{SKt}$ for classical tense logic is similar to $\mathbf{S_{BBI}}$ in that it has two residual rules corresponding to the traverse rules of $\mathbf{S_{BBI}}$. The main difference is that $\mathbf{SKt}$ uses only a tree structure of sequents and has no rule for associativity (which changes parent-child and sibling relations between sequents). Hence it is much easier to embed contraction rules in $\mathbf{SKt}$ than in $\mathbf{S_{BBI}}$ and the problem of search space explosion due to structural rules as in $\mathbf{CS_{BBI}}$ does not exist in $\mathbf{SKt}$.

9.3 Comparison between $\mathbf{S_{BBI}}$ and $\mathbf{DL_{BBI}}$

We have seen in Section 6.3 that for Boolean BI, sequents in $\mathbf{S_{BBI}}$ essentially represent a normal form of consecutions in $\mathbf{DL_{BBI}}$. For intuitionistic BI, Brotherston [10] establishes a stronger result that sequents in its sequent calculus are literally a normal form of consecutions in its display calculus and thus belong to the same syntactic category. He also conjectures that a cut-free sequent calculus for Boolean BI is unlikely to exist if it has no negative structural connective such as ♯ in $\mathbf{DL_{BBI}}$ (see Section 5 in [10]). Our discovery of $\mathbf{S_{BBI}}$ does *not* contradict his conjecture because we can think of sequents in $\mathbf{S_{BBI}}$ as implicitly applying a negative structural connective to falsehood contexts.

What is equally important, however, is that introducing only a negative structural connective is not enough to achieve a cut-free sequent calculus for Boolean BI, which must use a graph structure of sequents in which a sequent may have multiple parent sequents. In the case of $\mathbf{DL_{BBI}}$, the linear structural connective $\multimap$ allows such a graph structure of sequents, but the purpose of introducing $\multimap$ is to obtain the display theorem which is essential to Belnap's proof of the cut elimination theorem for display calculi [1]. Similarly, even though it does not require such a graph structure, the display calculus for intuitionistic BI in [10] also has the same linear structural connective $\multimap$. In contrast, $\mathbf{S_{BBI}}$ introduces an adjoint pair $W\langle\!\langle W'\rangle\!\rangle$ for the sole purpose of allowing such a graph structure of sequents.

10. Conclusion

Despite its close connection with separation logic, Boolean BI has not received much attention from the proof search community. Such a lack of research, which is quite unusual considering the status of separation logic in the field of program verification, is perhaps due to the difficulty of finding a proof theory suitable for theorem proving. Our nested sequent calculus $\mathbf{S_{BBI}}$ as well as a theorem prover based on it may serve as a test bed for developing proof search strategies for Boolean BI. In particular, its use of nested sequents allowing multiple parent sequents may shed new light on how to deal with separating implication in a theorem prover for separation logic.

Acknowledgments

We are grateful to the anonymous reviewers for their helpful comments. This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology (MEST) / National Research Foundation of Korea (NRF) (Grant 2012-0000472) and Mid-career Researcher Program through NRF funded by the MEST (2010-0022061).

References

[1] N. Belnap. Display logic. *Journal of Philosophical Logic*, 11:375–417, 1982.

[2] N. Belnap. Linear logic displayed. *Notre Dame Journal of Formal Logic*, 31:14–25, 1990.

[3] J. Berdine, C. Calcagno, and P. W. O'Hearn. A decidable fragment of separation logic. In *Proc. FSTTCS*, pages 97–109, 2004.

[4] J. Berdine, C. Calcagno, and P. W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *Proc. FMCO*, pages 115–137, 2005.

[5] J. Berdine, C. Calcagno, and P. W. O'Hearn. Symbolic execution with separation logic. In *Proc. APLAS*, pages 52–68, 2005.

[6] J. Berdine, C. Calcagno, B. Cook, D. Distefano, P. W. O'Hearn, T. Wies, and H. Yang. Shape analysis for composite data structures. In *Proc. CAV*, pages 178–192, 2007.

[7] L. Birkedal, N. Torp-Smith, and J. C. Reynolds. Local reasoning about a copying garbage collector. In *Proc. POPL*, pages 220–231, 2004.

[8] R. Brochenin, S. Demri, and É. Lozes. On the almighty wand. In *Proc. CSL*, pages 323–338, 2008.

[9] J. Brotherston. A cut-free proof theory for boolean BI (via display logic). Technical Report DTR09-13, Imperial College London, 2009.

[10] J. Brotherston. A unified display proof theory for bunched logic. In *Proc. MFPS*, pages 197–211, 2010.

[11] J. Brotherston and M. Kanovich. Undecidability of propositional separation logic and its neighbours. In *Proc. LICS*, pages 130–139, 2010.

[12] K. Brünnler. Deep sequent systems for modal logic. In *Proc. Advances in Modal Logic*, pages 107–119, 2006.

[13] B.-Y. E. Chang and X. Rival. Relational inductive shape analysis. In *Proc. POPL*, pages 247–260, 2008.

[14] D. Distefano and M. J. Parkinson. jStar: towards practical verification for Java. In *Proc. OOPSLA*, pages 213–226, 2008.

[15] D. Distefano, P. W. O'Hearn, and H. Yang. A local shape analysis based on separation logic. In *Proc. TACAS*, pages 287–302, 2006.

[16] K. Donnelly, T. Gibson, N. Krishnaswami, S. Magill, and S. Park. The inverse method for the logic of bunched implications. In *Proc. LPAR*, pages 466–480, 2004.

[17] D. Galmiche and D. Larchey-Wendling. Expressivity properties of boolean BI through relational models. In *Proc. FSTTCS*, pages 357–368, 2006.

[18] D. Galmiche and D. Méry. Proof-search and countermodel generation in propositional BI logic. In *Proc. TACS*, pages 263–282, 2001.

[19] D. Galmiche and D. Méry. Semantic labelled tableaux for propositional BI (without bottom). *Journal of Logic and Computation*, 13: 70–753, 2003.

[20] D. Galmiche and D. Méry. Tableaux and resource graphs for separation logic. *Journal of Logic and Computation*, 20:189–231, 2010.

[21] D. Galmiche, D. Méry, and D. J. Pym. The semantics of BI and resource tableaux. *Mathematical Structures in Computer Science*, 15: 1033–1088, 2005.

[22] R. Goré, L. Postniece, and A. Tiu. Cut-elimination and proof-search for bi-intuitionistic logic using nested sequents. In *Proc. Advances in Modal Logic*, pages 43–66, 2008.

[23] R. Goré, L. Postniece, and A. Tiu. Taming displayed tense logics using nested sequents with deep inference. In *Proc. TABLEAUX*, pages 189–204, 2009.

[24] R. Goré, L. Postniece, and A. Tiu. On the correspondence between display postulates and deep inference in nested sequent calculi for tense logics. *Logical Methods in Computer Science*, 7:1–38, 2011.

[25] S. S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. POPL*, pages 14–26, 2001.

[26] B. Jacobs, J. Smans, and F. Piessens. VeriFast: Imperative programs as proofs. In *Proc. VSTTE*, pages 59–68, 2010.

[27] R. Kashima. Cut-free sequent calculi for some tense logics. *Studia Logica*, 53(1):119–136, 1994.

[28] D. Larchey-Wendling and D. Galmiche. Exploring the relation between intuitionistic BI and boolean BI: an unexpected embedding. *Mathematical Structures in Computer Science*, 19:435–500, 2009.

[29] D. Larchey-Wendling and D. Galmiche. The undecidability of boolean BI through phase semantics. In *Proc. LICS*, pages 140–149, 2010.

[30] S. Magill, J. Berdine, E. M. Clarke, and B. Cook. Arithmetic strengthening for shape analysis. In *Proc. SAS*, pages 419–436, 2007.

[31] N. Marti, R. Affeldt, and A. Yonezawa. Formal verification of the heap manager of an operating system using separation logic. In *Proc. ICFEM*, pages 400–419, 2006.

[32] J. A. Navarro Pérez and A. Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *Proc. PLDI*, pages 556–566. ACM, 2011.

[33] H. H. Nguyen and W.-N. Chin. Enhancing program verification with lemmas. In *Proc. CAV*, pages 355–369, 2008.

[34] P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5:215–244, 1999.

[35] D. J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*. Kluwer Academic Pub, 2002.

[36] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. LICS*, pages 55–74, 2002.

[37] V. Vafeiadis and M. J. Parkinson. A marriage of rely/guarantee and separation logic. In *Proc. CONCUR*, pages 256–271, 2007.

[38] H. Yang. An example of local reasoning in BI pointer logic: the Schorr-Waite graph marking algorithm. In *Proceedings of the 1st Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management*, pages 41–68, 2001.