An Interactive Prover for Bi-intuitionistic Logic

Jean-René Courtault, Didier Galmiche and Daniel Méry

Université de Lorraine - LORIA
Campus Scientifique BP 239
Vandœuvre-lès-Nancy, France

Abstract

In this paper we present an interactive prover for deciding formulas in propositional bi-intuitionistic logic (BiInt). This tool is based on a recent connection-based characterization of bi-intuitionistic validity through bi-intuitionistic resource graphs (biRG). After giving the main concepts and principles we illustrate how to use this interactive proof or counter-model building assistant and emphasize the interest of bi-intuitionistic resource graphs for proving or refuting BiInt formulas. We complete this work by studying how to make our tool a fully automated theorem prover.

1 Introduction

Bi-intuitionistic logic (BiInt) is a conservative extension of intuitionistic logic that introduces a connective $\prec$, called co-implication, which acts as a dual to implication. It was first studied by Rauszer who gives a Hilbert calculus with Kripke and algebraic semantics [5] and more recently by Crolard as a way to define proof systems that work as programming languages in which values and continuations are handled in a symmetric way [1].

Cut-free calculi for BiInt have been proposed using deep inference, nested sequents [3] or labelled calculi [4] and a BDD-based approach has been recently developed [6]. In this paper we present an interactive proof-assistant that we view as a privileged way to become acquainted with our recently proposed connection-based characterization of bi-intuitionistic validity through R-Graphs [2]. In addition we analyse how to make this tool a fully automated theorem prover

2 Bi-intuitionistic Propositional Logic

Formulas of BiInt are built from a denumerable set $\mathcal{V} = \{\mathrm{P}, \mathrm{Q} \ldots\}$ of propositional letters w.r.t.the following grammar:

$$\mathrm{A} ::= \mathcal{V} \mid \bot \mid \mathrm{A} \vee \mathrm{A} \mid \mathrm{A} \wedge \mathrm{A} \mid \mathrm{A} \rightarrow \mathrm{A} \mid \mathrm{A} \prec \mathrm{A}.$$

Negation $\neg\mathrm{A}$, co-negation $\sim\mathrm{A}$ are respectively defined as syntactic sugar for $\mathrm{A} \rightarrow \bot$ and $\top \prec \mathrm{A}$, where $\top$ is defined as a shorthand for $\bot \rightarrow \bot$.

The Kripke semantics of BiInt is a straightforward extension of that of intuitionistic logic. A *Kripke model* $\mathcal{M}$ is a triple $\langle M, \sqsubseteq, [\![\cdot]\!]\rangle$, where M is a non-empty set of worlds, $\sqsubseteq$ is a partial order on M and $[\![\cdot]\!]$ is a function from worlds to sets of propositional letters satisfying the following *Kripke monotonicity* condition:

$$(\forall \mathrm{P} \in \mathcal{V})(\forall m \in M)(\forall n \in M)(\text{if } \mathrm{P} \in [\![m]\!] \text{ and } m \sqsubseteq n \text{ then } \mathrm{P} \in [\![n]\!]).$$

The *Kripke forcing relation* $\models$ associated with a Kripke model $\mathcal{M}$ is defined as the least relation between the worlds of $\mathcal{M}$ and formulas such that:

Figure 1: bi-intuitionistic R-graph

- $m \models \bot$ never;
- $m \models \mathrm{P}$ iff $\mathrm{P} \in [\![m]\!]$;
- $m \models \mathrm{A} \vee \mathrm{B}$ iff $m \models \mathrm{A}$ or $m \models \mathrm{B}$;
- $m \models \mathrm{A} \wedge \mathrm{B}$ iff $m \models \mathrm{A}$ and $m \models \mathrm{B}$;
- $m \models \mathrm{A} \to \mathrm{B}$ iff for all $n \in M$ such that $m \sqsubseteq n$, $n \not\models \mathrm{A}$ or $n \models \mathrm{B}$;
- $m \models \mathrm{A} \prec \mathrm{B}$ iff for some $n \in M$ such that $n \sqsubseteq m$, $n \models \mathrm{A}$ and $n \not\models \mathrm{B}$.

A formula C is *satisfied in* $\mathcal{M}$ iff $m \models \mathrm{C}$ for all worlds m in M and *valid* if it is satisfied in all Kripke models.

3 Bi-intuitionistic R-graphs

Let us recall some terminology. A *signed formula* is formula of BiInt paired with either a positive or negative *sign* ("+" or "−").

A *bi-intuitionistic R-graph* (biRG) is a directed multi-graph $G(V, E)$ with vertices V and edges E. V is required to contain a distinguished vertex ϵ called the *ϵ-vertex*. Figure 1 gives an example of a biRG named G_1.

All vertices u in a biRG are associated with a set $\mathcal{F}(G, u)$ of signed formulas the elements of which are called the *tags* of u in G. For example, in the biRG G_1 of Figure 1, we have $\mathcal{F}(G_1, a_1) = \{\mathrm{P}^+, \mathrm{Q}^-\}$ so that P^+ and Q^- are called tags of a_1 in G_1. We shall see later how tags are associated with vertices through the reduction of an initial bi-intuitionistic formula. Vertices are named with symbols in $\Psi \cup \Phi$, where $\Psi = \{\epsilon\} \cup \{a_i \mid i \in \mathbb{N}\}$ and $\Phi = \{x_i \mid i \in \mathbb{N}\}$. We use the letters a and x to denote arbitrary elements of Ψ and Φ. A ψ-vertex (ϕ-vertex) is a vertex named after a symbol in Ψ (Φ). V^Ψ (V^Φ) denotes the set of ψ-vertices (ϕ-vertices). ψ- and ϕ-vertices shall respectively play the role of constants and variables w.r.t. particular substitutions that are crucial to the forthcoming characterization of bi-intuitionistic validity.

In a biRG, all edges e are tagged with a letter $\mathcal{T}(G, e)$ from the set $\mathcal{T} = \{\psi, \phi, \bar{\psi}, \bar{\phi}, \sigma, \kappa\}$ of edge-tags. We write $u[\tau]v$ to denote the edge, tagged with the letter $\tau \in \mathcal{T}$, that goes from u to v and call this edge a τ-edge the source and target of which respectively are u and v. For example, in the biRG G_1 of Figure 1, $a_0[\phi]x_0$ is the ϕ-edge the source and target of which respectively are a_0 and x_0. Let us remark that since biRGs are multi-graphs, we are allowed to put more than one edge between two vertices as long as all edges have distinct edge-tags.

A biRG is *pure* if it does not contain any κ- or σ-edges. The pure part of a biRG G is the biRG obtained from G by removing all of its κ- and σ-edges.

Figure 2: bi-intuitionistic R-graphs in IBiS

A biRG is required to satisfy the following structural conditions:

- every ψ-edge ($\bar{\psi}$-edge) has a ψ-vertex as its target (source);
- every ϕ-edge ($\bar{\phi}$-edge) has a ϕ-vertex as its target (source);
- every σ-edge is a "bidirectional" link between a ϕ-vertex and a ψ-vertex such that either $u \in V^{\Psi}$ and $v \in V^{\Phi}$, or $u \in V^{\Phi}$ and $v \in V^{\Psi}$;
- every κ-edge $u[\kappa]v$ is a link between two vertices such that there is at least one formula occurring positively in $\mathcal{F}(G, u)$ and negatively in $\mathcal{F}(G, v)$.

Figure 2 gives examples of biRGs in IBiS (Interactive Bi-intuitionistic Solver). The name of a biRG is indicated along with its initial "root" vertex ϵ on the left hand side of a colon. Edges of type ϕ, $\bar{\phi}$ and κ (ψ, $\bar{\psi}$ and σ) are implemented as dotted (solid) arrows. ϕ- and ψ-edges have simple arrow-tips ($\rightarrow$), $\bar{\phi}$- and $\bar{\psi}$-edges have triangle arrow-tips ($\rightarrowtriangle$), κ-links have circle arrow-tips ($\multimap$) and σ-links have no arrow-tips.

Although in full generality one could construct arbitrary biRGs, we should restrict ourselves in this paper to biRGs that are generated by the decomposition of an initial bi-intuitionistic formula to prove (or disprove) through a reduction process. The details of this reduction process shall be explained in full details in Section 4.1 but for a quick intuitive understanding of the ideas underlying the notion, one should think of a biRG as a structure representing an initial sequent in a sequent-style derivation, or an atomic path in a connection-based setting.

biRGs are structures that convey both syntactic and semantic information and can therefore be interpreted either syntactically, or semantically. For example, an edge $u[\tau]v$, with $\tau \in \{\psi, \phi\}$ (intuitionistic edges) has a double intuitive meaning. From a semantic standpoint, $u[\tau]v$ means that, whenever interpreted in a Kripke model $\mathcal{M}$, the world corresponding to v should be accessible in $\mathcal{M}$ from the world corresponding to u, as prescribed by the Kripke interpretation of implication. Therefore, in the semantic interpretation, one reasons in terms of Kripke model accessibility. From a syntactic standpoint, in a sequent-style derivation for example, $u[\tau]v$ means that the formula $f(v)$ that led to the introduction of the vertex v should be introduced

Figure 3: initial R-graph for $(((\mathrm{R} \vee \mathrm{P}) \prec \mathrm{Q}) \wedge (\mathrm{P} \to \bot)) \to ((\mathrm{P} \to \mathrm{R}) \prec \mathrm{Q})$

before the formula $f(u)$ that led to the introduction of the vertex u. Therefore, in the syntactic interpretation, one reasons in terms of introduction precedence in sequent-style derivations.

For edges with tags in the set $\{\bar{\psi}, \bar{\phi}\}$ (bi-intuitionistic edges), the same semantic interpretation holds, but the syntactic interpretation breaks down (or, more precisely, needs to be reversed as indicated by the overbars on the edge-tags). Indeed, although oriented from u to v, the edge $u[\bar{\psi}]v$ implies that $f(u)$ is a subformula of $f(v)$ and thus has to be introduced after $f(v)$ in a sequent-style derivation. In order to recover the syntactic interpretation, one has to cross $\bar{\psi}$- and $\bar{\phi}$-edges backward (from their target to their source) when reasoning in terms of introduction precedence.

4 Interactive Bi-intuitionistic Solver (IBiS)

When IBiS[1] is started, the initial step consists in choosing the logic to consider in the area `"Choose Logic"`. Let us choose bi-intuitionistic propositional logic instead of plain intuitionistic propositional logic. Then, we need to enter the formula we want to prove in the area called `"Sequent or formula to prove"`. The help button helps us recall the right syntax for writing formulas. Let us try to prove the formula

$$\mathrm{C} = (((\mathrm{R} \vee \mathrm{P}) \prec \mathrm{Q}) \wedge (\mathrm{P} \to \bot)) \to ((\mathrm{P} \to \mathrm{R}) \prec \mathrm{Q})$$

as a first example. After pressing the validate button, the area called `"Current panel"` gets populated with the initial biRG G_0 depicted in Figure 3.

4.1 Reduction Step

Let us now explain how to build biRGs from (signed) formulas. A reduction through a formula A is a sequence $\mathcal{R} = R_0\rho_1 \ldots \rho_{i-1}\rho_i \ldots$ in which R_0 is a collection (of biRGs) containing the single biRG $G_0(V_0, E_0)$, where $V_0 = \{\epsilon\}$, $E_0 = \emptyset$ and $\mathcal{F}(G_0, \epsilon) = \{\mathrm{A}^-\}$, and each ρ_i is a reduction step that transforms the collection R_{i-1} inherited from the previous reduction step into a new

[1] Available at `http://www.loria.fr/~galmiche/TYPES.html`

α	α_1	α_2	β	β_1	β_2
$(\mathrm{A} \wedge \mathrm{B})^+$	A^+	B^+	$(\mathrm{A} \wedge \mathrm{B})^-$	A^-	B^-
$(\mathrm{A} \vee \mathrm{B})^-$	A^-	B^-	$(\mathrm{A} \vee \mathrm{B})^+$	A^+	B^+
$(\mathrm{A} \rightarrow \mathrm{B})^-$	A^+	B^-	$(\mathrm{A} \rightarrow \mathrm{B})^+$	A^-	B^+
$(\mathrm{A} \prec \mathrm{B})^+$	A^+	B^-	$(\mathrm{A} \prec \mathrm{B})^-$	A^-	B^+

	itype
$(\mathrm{A} \rightarrow \mathrm{B})^+$	ϕ
$(\mathrm{A} \rightarrow \mathrm{B})^-$	ψ
$(\mathrm{A} \prec \mathrm{B})^+$	$\overline{\psi}$
$(\mathrm{A} \prec \mathrm{B})^-$	$\overline{\phi}$

Figure 4: Reduction rules w.r.t. principal and intuitionistic types

Figure 5: pure irreducible biRGs through $(((\mathrm{R} \vee \mathrm{P}) \prec \mathrm{Q}) \wedge (\mathrm{P} \rightarrow \bot)) \rightarrow ((\mathrm{P} \rightarrow \mathrm{R}) \prec \mathrm{Q})$

collection R_i by applying one of the reduction rules given in Figure 4 to a signed formula tagging a vertex u of some biRG in R_{i-1}.

Reduction rules rewrite biRGs according to the principal type (α or β) and intuitionistic type (ϕ, $\overline{\phi}$, ψ or $\overline{\psi}$) of the formula under reduction. β-rules split a biRG into two new biRGs, ϕ- and $\overline{\phi}$-rules introduce new ϕ-vertices, while ψ- and $\overline{\psi}$-rules introduce new ψ-vertices. $\beta\phi$- and $\beta\overline{\phi}$-formulas are special in that they may be reduced as many times as required to decide validity (duplication is handled through the notion of multiplicity in [2]).

A biRG is *irreducible* if all of its non atomic formulas have been reduced at least once and a collection of biRGs is *irreducible* if all of its biRGs are irreducible. Let $\mathcal{R}$ be a reduction through A, $\mathcal{R}$ is *finished* if there exists a natural number n such that for all $R_i \in \mathcal{R}$, R_i is irreducible if and only if $i \geq n$ and we then call R_n a pure irreducible collection of biRGs through A.

In IBiS, starting with the initial biRG G_0, we now have two possibilities: either we choose to apply reduction rules step by step manually by right-clicking on vertices and selecting a formula to reduce in the combo box "`Reduction rules`" (only reducible formulas are listed in the box), or we choose to apply all possible reduction rules automatically in one go by pressing the hat button. Let us press the hat button, after the reduction is finished, the current panel gets populated with the pure irreducible biRGs through $(((\mathrm{R} \vee \mathrm{P}) \prec \mathrm{Q}) \wedge (\mathrm{P} \to \bot)) \to ((\mathrm{P} \to \mathrm{R}) \prec \mathrm{Q})$ depicted in Figure 5.

4.2 Binding Step

The next step consists in applying binding rules, *i.e.*, adding σ- and κ-edges to the collection R_f of irreducible biRGs obtained in the reduction step. To insert a κ-edge into a biRG, one needs to right-click on the vertex which in supposed to become the source of the κ-edge, then click on "`Binding rules`", then select the "`Kappa F+ -> F-`" action, which unfolds a list of vertices from which we should select the one that should become the target of the κ-edge.

Selecting the "`Sigma xi -> ai`" action in the "`Binding rules`" list allows one to insert σ-edges in a similar way except that any newly inserted σ-edge is automatically added in all biRGs in which it can be inserted, *i.e.*, once a variable x is bound to constant a by a σ-edge in some biRG, it is also bound to that same constant in all biRGs that contains both x and a. Let us remark that the "`History`" panel allows one to rollback to any previous state by clicking on one of the undo buttons.

4.3 Validity Checking Step

Before describing this step in IBiS we need to recall the biRG characterization of BiInt validity [2] and the notions upon which it relies.

A T-path from u to v in a biRG $G(V, E)$ is a sequence $u_0\tau_1u_1 \ldots u_{p-1}\tau_pu_p$ such that $u_0 = u$, $u_p = v$ and for all $1 \leqslant i \leqslant p$, u_i is a vertex in V, τ_i is an edge-tag in T and E contains a τ_i-edge $u_{i-1}[\tau_i]u_i$. A T-path is called a concrete path whenever $T = \{\psi, \bar{\psi}, \sigma\}$. T-cycles and concrete cycles are defined accordingly. Concrete reachability is defined as the relation $Gu \to v$ that holds if and only if $u = v$ or there is a concrete path from u to v in G.

Let $G(V, E)$ be a biRG. G is *inconsistent* if there is some vertex u in V such that $\bot^+ \in \mathcal{F}(G, u)$. G is *admissible* if all concrete cycles in G are σ-cycles that contain at most one ψ-vertex and for all τ-edges $u[\tau]v$ in E such that $\tau \in \{\phi, \bar{\phi}\}$, $Gu \to v$. A collection of biRGs is inconsistent (admissible) if it contains an inconsistent (admissible) biRG. Given a subset S of V, the *S-slice* of G is defined as the smallest biRG $G^S(V^S, E^S)$ such that $S \subseteq V^S$ and for all $u \in V^S$ and $v \in V$, if $v[\tau]u \in E$ and $\tau \in \{\psi, \phi, \sigma\}$ then $v \in V^S$ and $v[\tau]u \in E^S$, and if $u[\tau]v \in E$ and $\tau \in \{\bar{\psi}, \bar{\phi}, \sigma\}$ then $v \in V^S$ and $u[\tau]v \in E^S$. G is *complementary* if there is a κ-edge $u[\kappa]v$ in E such that the slice $G^{\{u,v\}}$ is admissible and $G^{\{u,v\}}u \to v$. A collection is complementary if all of its biRGs are complementary.

Let $G(V, E)$ be a biRG. The pure part of G induces a relation $\ll$ on $V \times V$, called the *domination ordering*, defined as the least partial ordering such that $u \ll v$ if $u[\phi]v \in E$ or $u[\psi]v \in E$ or $v[\bar{\phi}]u \in E$ or $v[\bar{\psi}]u \in E$. A local σ-binding for G is a function σ that extends G by adding σ-links (bidirectional σ-edges) in E. Given two collections $R = \{G_1, \ldots, G_n\}$ and $S = \{\sigma_1, \ldots, \sigma_n\}$ such that for all $1 \leqslant i \leqslant n$, σ_i is a local σ-binding for G_i, the global σ-binding $\overline{\sigma}$ for R (induced by S) is defined as $\overline{\sigma}(G_i) = \sigma_i(G_i)$ for all $1 \leqslant i \leqslant n$.

A global σ-binding induces a relation $\sqsubset$ on $\Psi \times \Phi$, called a *precedence relation*, such that $a \sqsubset x$ if there is a σ-link $a[\sigma]x$ in $\overline{\sigma}(G_i)$ for some $1 \leqslant i \leqslant n$. A local σ-binding σ is admissible for a biRG G if $\sigma(G)$ is admissible. A global σ-binding $\overline{\sigma}$ is admissible for a biRG-collection R

if for all G in R, $\overline{\sigma}(G)$ is admissible. Local and global κ-bindings are defined similarly w.r.t. κ-edges.

Definition 1. *A* BiInt *formula* A *is biRG-valid if and only if there exist a (global) σ-binding $\overline{\sigma}$ and a (global) κ-binding $\overline{\kappa}$ for some collection R_f of pure irreducible biRGs through* A *such that:*

- *For all (not necessarily distinct) biRGs $G_1(V_1, E_1), G_2(V_2, E_2)$ in $\overline{\sigma} \circ \overline{\kappa}(R_f)$ and all ϕ-vertices x in $V_1 \cap V_2$, if $x[\sigma]u \in E_1$ and $x[\sigma]v \in E_2$ then $u = v$.*
- *For all consistent biRGs $G(V, E)$ in $\overline{\sigma} \circ \overline{\kappa}(R_f)$, $G(V, E)$ is complementary.*
- *The* reduction ordering $\lhd = (\ll \cup \sqsubset)^+$ *induced by $\overline{\sigma}$, where $(\cdot)^+$ stands for transitive closure, is irreflexive.*

The binding rules should be used so as to try to make the irreducible collection of biRGs satisfy either validity, or non-validity criteria. IBiS provides some assistance on that matter through its green and red player buttons (and). When the green player button is clicked, IBiS checks the collection of biRG in the current panel for validity, while the red player button checks it for non-validity and, upon success, returns a counter-model.

It is also possible to check whether a particular biRG in the current panel individually (or locally) satisfies the validity or non-validity criteria. For that, one needs to right-click on the biRG that should be checked and select either the `"Check validity"`, or the `"Check non validity"` action. Let us remark that Definition 1 implies that one only needs to consider consistent biRGs to establish the biRG-validity of a formula. Therefore, inconsistent biRGs are by default hidden in IBiS, but checking `"Show inconsistent graphs"` in the `"Options"` area makes them visible.

In the collection R_f given in Figure 2, all biRGs in R_f are admissible and complementary. Moreover, R_f induces the following acyclic reduction ordering:

After we click on the green player button to check the validity criteria on R_f, IBiS concludes to the validity of the formula C we entered in the initial step.

Figure 6 gives an example of counter-model construction for the formula $\mathrm{Q} \to ((\neg(\mathrm{P} \prec \mathrm{Q}) \prec \mathrm{P}) \vee \mathrm{P})$ by clicking on the red player button on a collection of saturated (see [2] for a definition of saturation) biRGs where all possible κ-edges have been inserted.

5 Towards proof-search automation

To make IBiS a fully automated theorem prover one needs to solve two distinct problems. The first one is to provide an algorithm to enumerate all possible global κ-bindings. In a connection-based or matrix-based setting, this would roughly correspond to a path enumerating algorithm. The second problem is to provide an algorithm to find admissible global σ-binding.

Given an admissible σ-binding, it is not difficult to check whether κ-edges are covered by a concrete path or not, the actual problem really is to find such σ-bindings. A trivial but

Figure 6: counter-model for $Q \to ((\neg(P \prec Q) \prec P) \vee P)$

particularly inefficient solution would be to enumerate all possible σ-bindings and check whether they are admissible or not.

As a first step toward more efficient solutions, in [2], we briefly sketched an algorithm that, given a pure biRG G and a set K of κ-edges, enumerates only those σ-bindings that are admissible for the slice of G induced by K. The solving algorithm proceeds by preserving the acyclicity of the underlying reduction ordering $\lhd$ and has been implemented in SWI-Prolog for a first prototype [2].

Figure 10 shows how to run the Prolog prototype in an interactive SWI-Prolog session. We first run the SWI-Prolog interpreter from the terminal with the command `swipl`. Once in the Prolog interpreter, we get a question mark prompt from which we load the source code of the solver prototype contained in a file called `iwil.pl`. We are now ready to manipulate resource graphs. Our first predicate (command) is `readrg` which loads a resource graph description from a file the name of which defaults to `rg.pl`.
The content of the `rg.pl` file used in Figure 10 is given in Figure 7 as well as its more pleasant graphical counterpart. We use a binary predicate `rg` to define one single resource graph named `iwil` by enumerating all edges that should belong to it. This single 'global' resource graph is obtained by merging the pure part of the resource graphs in the collection shown in Figure 2, keeping exactly one κ-edge in each consistent resource graph of the collection [3].

Since `rg.pl` may contain more than one resource graph description, we need to indicate which resource graph should be used as the working resource graph. This is done by issuing the predicate `userg(iwil)`, which tells the solver to use the resource graph named `iwil` in `rg.pl`. We can check that the working resource graph has been correctly set by issuing the predicate

[2] Available at `http://www.loria.fr/~galmiche/TYPES.html`
[3] In a connection-based setting, this would correspond to a spanning set of connections.

```
root@nout: ~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
meryd@shou:~/Logiciel/TYPES/IWIL$ cat rg.pl
rg(iwil,edge(psi,0,a0)).
rg(iwil,edge(psi,x1,a2)).
rg(iwil,edge(psi,x1,a3)).
rg(iwil,edge(phi,a0,x0)).
rg(iwil,edge(isp,a1,a0)).
rg(iwil,edge(ihp,x1,a0)).
rg(iwil,edge(kap,a1,a3)).
rg(iwil,edge(kap,a1,x0)).
rg(iwil,edge(kap,x1,a1)).
meryd@shou:~/Logiciel/TYPES/IWIL$
```

Figure 7: Prolog description of the R-graph `iwil`

Figure 8: working slice of the R-graph `iwil`

`printrg`, which prints the working resource graph.

Since the working graph contains κ-edges, the next step is to compute the slice induced by the set of κ-edges. This is achieved by issuing the `slicerg` predicate as illustrated in Figure 11. We can check that the slice has been computed correctly with the `printslice` predicate, which prints the working slice. Computing the working slice tells us that the vertex a_2 is not needed under the κ-binding depicted in Figure 7 and can therefore be safely discarded. The resulting slice is graphically depicted in Figure 8.

Once we have a working slice, the next step is to ask the solver to enumerate all admissible σ-bindings. We do so with the `solvemap` predicate. When the solver outputs a solution, we can either stop the solving process by typing return or get the next solution (if any) by typing a semi-colon. In that case, the `solvemap` predicate fails when all admissible solutions have been exhausted. In our example, we can see that there exists only one solution which exactly corresponds to the σ-binding $\{x_1[\sigma]a_1, x_0[\sigma]a_0\}$ used as a solution in Figure 2. We can apply this σ-binding to the working slice with the `applymap` predicate and see the result with `printslice` as illustrated in Figure 11. The resulting slice is depicted in Figure 9. Finally, to convince

Figure 9: admissible σ-binding for the working slice of the R-graph `iwil`

ourselves that the solution given by the solver actually is a solution, we check that all non-solid (virtual) edges are indeed covered by a concrete path using the `checkcpath` predicate [4] as illustrated in Figure 12.

6 Future Work

We have explained in the previous section how to find σ-bindings using our Prolog solver prototype. A Java rewrite of this Prolog prototype is currently in progress to enable fully automated σ-binding resolution in IBiS. The task is not as direct as it may seem at first glance. Indeed, because we merge a whole resource graph collection in which each resource graph corresponds to an initial sequent in a sequent-style derivation (or an atomic path in a connection-based setting) into one single global resource graph, the solutions given by the `solvemap` predicate should be checked further in order to ensure that they still work on each local resource graph in the initial resource graph collection. We conjecture that it is possible to do without this additional check for local correction for resource graph collections generated by reduction of an initial signed formula as prescribed by the reduction rules of Figure 4.

References

[1] T. Crolard. A formulae-as-types interpretation of substractive logic. *Journal of Logic and Computation*, 14(4):529–570, 2004.

[2] D. Galmiche and D. Méry. A Connection-based Characterization of Bi-intuitionistic Validity, In *23rd Int. Conference on Automated Deduction, CADE-23, LNAI 6803*, pages 268–282, Wroclaw, Poland, 2011. Springer Verlag.

[3] R. Goré, L. Postniece, and A. Tiu. Cut-elimination and proof-search for Bi-intuitionistic logic using nested sequents. In *Advances in Modal Logic 7*, pages 43–66, 2008. College Publications, London.

[4] L. Pinto and T. Uustalu. Proof search and counter-model construction for Bi-intuitionistic propositional logic with labelled sequents. In *Int. Conference on Analytic Tableaux and Related Methods, TABLEAUX 2009, LNAI 5607*, pages 295–309, Oslo, Norway, 2009. Springer Verlag.

[5] C. Rauszer. An algebraic and Kripke-style approach to a certain extension of intuitionistic logic. *Dissertationes Mathematicae*, 168, 1980.

[6] R. Goré, and J. Thomson. BDD-Based Automated Reasoning for Propositional Bi-intuitionistic Tense Logics. In *6th Int. Joint Conference on Automated Reasoning, IJCAR 2012, LNAI 7364*, pages 301–315, Manchester, UK, 2011. Springer Verlag.

[4] Failure of the `checkcpath` under a solution given by the solver would expose a bug in the implementation of the solver.

```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
meryd@shou:~/Logiciel/TYPES/IWIL$ swipl
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 3,856 bytes
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 5.10.4)
Copyright (c) 1990-2011 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [iwil].
% iwil compiled 0.01 sec, 64,112 bytes
true.

?- readrg.
Define: rg(iwil,edge(psi,0,a0))
Define: rg(iwil,edge(psi,x1,a2))
Define: rg(iwil,edge(psi,x1,a3))
Define: rg(iwil,edge(phi,a0,x0))
Define: rg(iwil,edge(isp,a1,a0))
Define: rg(iwil,edge(ihp,x1,a0))
Define: rg(iwil,edge(kap,a1,a3))
Define: rg(iwil,edge(kap,a1,x0))
Define: rg(iwil,edge(kap,x1,a1))
true.

?- userg(iwil).
true.

?- printrg.
x1.[kap].a1
a1.[kap].x0
a1.[kap].a3
x1.[ihp].a0
a1.[isp].a0
a0.[phi].x0
x1.[psi].a3
x1.[psi].a2
0.[psi].a0
true.
```

Figure 10: loading an R-graph into the Prolog solver prototype

Figure 11: finding a solution with the Prolog solver prototype

```
?- solvemap(Map).
Map = [x1/a1, x0/a0] ;
false.

?- applymap([x1/a1, x0/a0]).
true.

?- printslice.
x1.[kap].a1
x1.[ihp].a0
a1.[kap].x0
a1.[kap].a3
a1.[isp].a0
0.[psi].a0
a0.[phi].x0
x1.[psi].a3
x1.[sig].a1
a1.[sig].x1
x0.[sig].a0
a0.[sig].x0
true.

?- setecho(debug).
true.

?- checkcpath(edge(ihp,x1,a0)).
  virtual edge:  x1.[ihp].a0
  concrete path: x1.[sig].a1.[isp].a0
true.

?- checkcpath(edge(phi,a0,x0)).
  virtual edge:  a0.[phi].x0
  concrete path: a0.[sig].x0
true.

?- erasemap([x1/a1, x0/a0]).
true.

?-
```

Figure 12: checking a solution with the Prolog solver prototype