Available online at [www.sciencedirect.com](http://www.sciencedirect.com/)

[Electronic Notes in Theoretical Computer Science 323 (2016) 215–233](http://dx.doi.org/10.1016/j.entcs.2016.06.014)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

On Graphs for Intuitionistic Modal Logics [1](#_bookmark1)

Paulo A. S. Veloso[2](#_bookmark0)

*Programa de Engenharia de Sistemas e Computa¸c˜ao, COPPE Universidade Federal do Rio de Janeiro (UFRJ)*

*Rio de Janeiro, RJ, Brazil*

Sheila R. M. Veloso[3](#_bookmark0)

*Departamento de Engenharia de Sistemas e Computa¸c˜ao, Fac. Engenharia Universidade do Estado do do Rio de Janeiro (UERJ)*

*Rio de Janeiro, RJ, Brazil*

**Abstract**

We present a graph approach to intuitionistic modal logics, which provides uniform formalisms for express- ing, analysing and comparing Kripke-like semantics. This approach uses the flexibility of graph calculi to express directly and intuitively possible-world semantics for intuitionistic modal logics. We illustrate the benefits of these ideas by applying them to some familiar cases of intuitionistic multi-modal semantics.

*Keywords:* Intuitionistic modal logics, semantics, graph formulations, calculi, refutation, special relations.

# Introduction

We present a graph approach to intuitionistic modal logics, which provides a flexible and uniform tool for expressing, analysing and comparing possible-world semantics. This graph approach can be regarded as a version of diagrammatic reasoning, where we can express formulas by diagrams, which can be manipulated to unveil properties (like consequence and satisfiability). Graph representations and transfor- mations, having precise syntax and semantics, give proof methods. An interesting feature of this graph approach is its 2-dimensional notation providing pictorial rep- resentations that support visual manipulations [[4](#_bookmark38)]. These ideas have been adapted

to refutational reasoning [[14](#_bookmark45)] and applied to multi-modal classical logics [[15](#_bookmark50)].

1 Research partly sponsored by the Brazilian agency CNPq. Helpful comments by the referees are gratefully acknowledged.

2 Email: [pasveloso@gmail.com](mailto:myuserid@mydept.myinst.myedu)

3 Email: [sheila.murgel.bridge@gmail.com](mailto:couserid@codept.coinst.coedu)

<http://dx.doi.org/10.1016/j.entcs.2016.06.014>

1571-0661/© 2016 The Author(s). Published by Elsevier B.V.

This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

Modal logics and graphs are closely connected. Kripke semantics can be pre- sented via directed labelled graphs for the accessibility relation of each modality [[2](#_bookmark37)].

It is natural to represent that *a* is related to *b* via relation *r* by an arrow *a r b*.

*−→*

Intuitionistic modal logic is an interesting subject [[6](#_bookmark40),[11](#_bookmark46)]. There seems to be little consensus on the appropriate approach to its semantics, as indicated by the diversity of Kripke-like semantics proposed (see [[5](#_bookmark41),[13](#_bookmark47)] and references therein).

We provide graph calculi, having diagrams as terms and whose rules transform diagrams, capturing graphically the semantics of the modal operators and accessi- bility relations. These calculi provide uniform and flexible formalisms where one can explore Kripe-like semantics for intuitionistic modal logics: satisfaction conditions, valid formulas, etc. We illustrate these ideas by 2 case studies: logics as in [[13](#_bookmark47),[5](#_bookmark41)].

We will consider a modal language ML, with set Φ of formulas, given by sets PL, of propositional letters, and RS, of 2-ary relation symbols. The formulas of ML are generated by the grammar *ϕ* ::= *⊥|* p *| ϕj ∧ ϕjj | ϕj ∨ ϕjj | ϕj → ϕjj | ⟨*r*⟩ϕ |* [r]*ϕ*. [4](#_bookmark4)

# Graphs and Modalities: Basic Ideas

We now introduce informally some basic ideas about graphs and modalities. [5](#_bookmark5)

A graph amounts to a finite set of (alternative) slices. A slice S consists of an

underlying draft S together with a distinguished node (marked, e. g. w^ ). A draft

amounts to finite sets of nodes and arcs. Slices and graphs represent sets of states, whereas drafts (and sketches, see Section [3](#_bookmark7)) will describe restrictions on states.

Arcs may be binary or unary. A binary arc stands for accessibility between states; we represent that node v is accessible from node u by the relation of r by

a solid arrow labelled r from u to v: u r v (abbreviated u r v). A unary arc is

*−→*

meant to capture the fact that a formula holds at a state; we represent that formula

*ϕ* holds at node w by a dashed line from w to *ϕ*: w ,˛*ϕ* (abbreviated w*|ϕ*).

Expressions will encompass slices, graphs and their complements (noted by an overbar). As such, an expression represents a set of states; so we can also use unary arcs of the form w ,˛E , where E is an expression.

We now introduce some concepts to be used and illustrated in Example [2.1](#_bookmark6).

A (draft) morphism is a node mapping that preserves arcs. A (slice) homomor- phism is a morphism of their underlying drafts that preserves distinguished nodes. A draft may have conflicts that prevent its satisfaction. We consider two kinds of conflicts. One concerns contradictory 1-ary arcs: if draft D has the pattern E rz w ,˛E , then expression E is a witness of a conflict at node w. If D has 1-ary arc w ,˛Q , slice Q will be a witness of a conflict at node w if there is a morphism from Q to D mapping the distinguished node of Q to w.

To reason about modal formulas, we convert them to expressions (with the same meaning) and reason graphically about these. We reduce consequence to unsatisfi- ablility: “every state satisfying *ψ*1, ..., *ψn* also satisfies *θ*” (noted *{ ψ*1*,..., ψn } |*=

4 As usual, *чϕ* abbreviates *ϕ → ⊥*.

5 These and other ideas will be formulated more precisely later on: in Section [3](#_bookmark7).

*θ*) is equivalent to “there is no state satisfying *ψ*1, . . . , *ψn* and failing to satisfy *θ*” (noted *{ ψ*1*,..., ψn, θ } |*= *⊥*). Notice that *θ* is not a formula: is complementation rather than intuitionistic negation. The next example illustrates this approach.

**Example 2.1** (Consequence via slice conversions) We reduce *⟨*r*⟩* (*ψ ∧ θ*) *|*= *⟨*r*⟩ ψ*

to *{ ⟨*r*⟩* (*ψ ∧ θ*)*, ⟨*r*⟩ ψ } |*= *⊥*. We first indicate the graph-calculus steps.

(*⟨ ⟩*)

w^

r /*¸* ,˛

z

*ψ*

` ˛¸ x

E

* 1. We convert *⟨*r*⟩ ψ* to expression E: *⟨*r*⟩ ψ ≈* .
  2. We also convert *⟨*r*⟩* (*ψ ∧ θ*) to slice P as follows:

(*⟨ ⟩*)

(*Λ*)

r

¸,*ψ*

*⟨*r*⟩* (*ψ ∧ θ*)

*≈* ^u r /v*¸* ,˛*ψ ∧ θ* ^u

,¸*θ*

/v*¸*

*≈*

` ˛¸ x

P

* 1. We now obtain, from P and E, the following slice P*j* (for *{⟨*r*⟩*(*ψ ∧ θ*)*, ⟨*r*⟩ψ}*):

¸,*ψ*

` ˛¸

*ψ* rz z ¸*,*r w^

E

x

z u^ r /v*¸*

r

` ˛¸

P

,¸*θ*

x

* 1. Call Q := w r /z *¸* ,˛*ψ* the slice under complement in E. So, slice P*j* is:

^

¸,*ψ*

Q rz

^u r /v*¸*

,¸*θ*

` ˛¸ x

P

We have a homomorphism *η* from Q to P (cf. p. [2](#_bookmark2)), given by w *'→* u, z *'→* v:

Q w r /z *¸*

^

*η η*

,˛*ψ*

P v*z* r /*¸* *s* ,˛

^u v

*ψ*

˛¸*θ*

Slice P*j* has conflict at node u, with slice witness Q (cf. p. [2](#_bookmark2)).

We now provide intuitive explanations for these steps, using R for the relation of r.

r /*¸* ,˛

1. The states pertaining to E are those not pertaining to slice w^ z *ψ* .

The states s pertaining to this slice Q are those for which there is a state t such that sRt and t satisfies the unary arc z ,˛*ψ* (i. e., t satisfies formula *ψ*).

1. The states s pertaining to P are those for which there is a state t such that

sRt and t satisfies both arcs v ,˛*ψ* and v ,˛*θ* (i. e., t satisfies *ψ ∧ θ*).

1. The states pertaining to slice P*j* are those pertaining to slice P that satisfy the unary arc u ,˛E = u ,˛Q.
2. Any state pertaining to P must (as *η* :Q *→* P) pertain to Q, whence it does not pertain to Q. Thus, there is no state pertaining to P*j*, so it is not satisfiable.

Hence, set *{⟨*r*⟩* (*ψ ∧ θ*)*, ⟨*r*⟩ ψ}* is unsatisfiable and *⟨*r*⟩* (*ψ ∧ θ*) *|*= *⟨*r*⟩ ψ*. *q*

# Graph Concepts and Results

We now introduce some basic concepts and results about graphs. [6](#_bookmark9) We will use an infinite set Nd of nodes; the *ﬁrst 3 nodes* being x, y and z.

A *graph language* GL is characterized by two sets of symbols: Sb1, of unary ones, and Sb2, of binary ones. Its syntax is defined by mutual recursion as follows.

(E) The *expressions* are the 1-ary symbols s *∈* Sb1, the slices and the graphs (see below), as well as E, for an expression E.

(a) The *arcs over* a set N *⊆* Nd of nodes are as follows.

(1) A unary arc w*|*E over N consists of a node w *∈* N and an expression E.

(2) A binary arc u L v over N consists of nodes u*,* v *∈* N and 2-ary symbol L *∈* Sb2.

(Σ) A *sketch* Σ= *⟨*N; A*⟩* consists of 2 sets: N *⊆* Nd (of nodes) and A of arcs over N.

(D) A *draft* is a sketch with finite sets of nodes and arcs.

(S) A *slice* S= *⟨*N; A : w*⟩* consists of an *underlying draft* S := *⟨*N; A*⟩* and a *distinguised node* w *∈* N. We often use the notation S = (S : w).

(G) A *graph* is a finite set of slices.

A *proper* sketch has non-empty node set. The *positive part* of a sketch consists of its nodes and its complement-free arcs. The *empty graph {}* has no slices.

A *structure* M for graph language GL consists of a universe M */*= *∅*, as well as a subset sM *⊆* M, for each s *∈* Sb1, and a binary relation LM on M, for each L *∈* Sb2.

We now define semantics also by mutual recursion.

**(**E**)** The *extension* [E]M of expression E is defined as follows. For a 1-ary symbol s *∈* Sb1: [s]M := sM; if E is a slice or a graph, then we use its behaviour: [E]M := [[[E]**]**M (see below); and, for E, we use complement: [E]M := M *\* [E]M.

**(**g**)** An *assignment* for N *⊆* Nd is a function g :N *→* M (so w *∈* N *'→* wg *∈* M).

**(**a**)** We define *satisfaction* (in M) for an arc over set N as follows.

(1) Assignment g satisfies unary arc w*|*E (in M) iff wg *∈* [E]M.

(2) Assignment g satisfies binary arc u L v (in M) iff (ug*,* vg) *∈* LM.

**(**Σ**)** Assignment g satisfies a sketch (in M) iff it satisfies all its arcs.

6 For more details about graphs see, e. g. [[14](#_bookmark45),[15](#_bookmark50)] and references therein.

**(**S**)** For a slice S = (S : w), its *behaviour* (in R) is the set [[S]**]**R consisting of the values wg *∈* M for the assignments g satisfying its underlying draft S.

**(**G**)** For a graph G, its *behaviour* (in R) is [[[G]**]**R := S*∈*G [[S]**]**R .

We define *satisﬁability*, *equivalence* and *nullity* as follows. Consider a class of models kk . A sketch Σ is satisfiable in kk iff there exist a model R *∈* kk and an assignment satisfying Σ in R. A slice S is satisfiable in kk iff its underlying draft S is so; and a graph G is satisfiable in kk iff some slice S *∈* G is so. Expressions E and F are equivalent in kk iff, for every model R *∈* kk: [E]R = [F]R. An expression set E is null in kk iff, for every model R *∈* kk: E*∈***E** [E]R = *∅*. We use simply *satisﬁable*, *equivalent* (noted *≡*) and *null* when referring to the class of all models. For instance, a singleton graph *{* S *}* and its slice S are equivalent (so we can identify them); the empty graph *{}* is null, as is the formula *⊥*.

We now define structural comparison and conflicts, introduced in Section [2](#_bookmark3).

For sketches Δ and Σ, a *morphism* from Δ to Σ is a function *μ* : NΔ *→* NΣ (noted *μ* : Δ −−· Σ), for which we have *μ*(a) *∈* AΣ, for every arc a *∈* AΔ (with *μ*(w*|*E) := w*μ|*E and *μ*(u L v) := u*μ* L v*μ*). Now, given slices Q = (Q **:** v) and P = (P : u), a *homomorphism* from Q to P is a function *η* : NQ *→* NP (noted *η* :Q *→* P) that is a morphism *η* : Q −−· P and v*η* = u.

Morphisms transfer satisfying assignments by composition. Given a morphism *μ* :Δ −−· Σ, if g satisfies Σ in R, then the composite g *· μ* satisfies Δ in R. Thus, if there exists a homomorphism *η* :Q *→* P, then [[Q]**]**R *⊇* [[P]**]**R .

Consider a sketch Σ = *⟨*N; A*⟩*. An expression E with u*|*E*,* u*|*E *∈* A is an *expression witness* of Σ at node u *∈* N. A slice Q = (Q **:** v) for which there is a morphism *μ* : Q −−· Σ such that v*μ|*Q *∈* A is a *slice witness* of Σ at node v*μ ∈* N. A sketch is *zero* iff it has some witness. A slice S is *zero* iff S is zero. A graph is *zero* iff all its slices are zero. Clearly, a zero sketch is not satisfiable; so zero slices and graphs are null. One can effectively decide whether a draft, a slice or a graph is zero.

We use ‘+’ for adding arcs (and their nodes). To glue a slice S on node w of slice P, we take a copy S*j* of S having only its distinguished node in common with P and add S*j* to P, thereby obtaining a *glued slice* PwS. We glue a graph by

gluing its slices: PwG := *{*PwS */* S *∈* G*}*. For instance, P := ^v L /w*¸* ,˛E and

S := u K /v*¸* ,˛F have PwS= v L /w*¸* K /v*∗¸* ,˛F as glued slice.

^ ^

,¸E

A proper sketch Σ = *⟨*N; A*⟩* gives a *natural structure* N[Σ]: with universe N, sN[Σ] := *{*w *∈* N */* w*|*s *∈* A*}* (s *∈* Sb1), LN[Σ] := *{*(u*,* v) *∈* N2 */* uLv *∈* A*}* (L *∈* Sb2).

**Example 3.1** (Natural construction) Consider the following draft D:

L

s rz x ,˛

^

L

^x L /y*¸* ,˛

^x L /y*¸* ,˛s

s rz yL /z *¸* ,˛s

The positive part D+ of D and the natural structure N[D] are as follows:

L LN[Σ]



x

x

L

s rz yL /z*¸* ,˛s

N[Σ] z

LN[Σ]

LN[Σ] /*¸*

,˛N[Σ]

s r y  z  s

Consider the identity assignment 1 on set *{*x*,* y*}*.

* 1. Assignment 1 satisfies the arcs of D+ as well as the 1-ary arc x*|*s.
  2. We can see that assignment 1 also satisfies the unary arc

x ,˛ ^x L /y*¸* ,˛ ^x L /y*¸* ,˛s .

Thus, assignment 1 satisfies draft D in natural structure N[D]. *q*

In the sequel, we will show how one can represent a modal formula by an expres- sion (of a graph language) with the same meaning, thus reducing questions about formulas to questions on expressions. We will be able to eliminate logical symbols from a modal formula, converting it to an equivalent expression. The elimination rules mimic the semantics of the modal language. The following ones are general.

We can eliminate double complement. We can also move complement inside by rules like De Morgan laws: *rule* (*∪*) converts a complemented graph G to the slice with arcs^x*|*S, for S *∈* G; for a slice S having only arcs of the form w^ *|*E*i*, *rule* (*∩*)

(*∪*)

(*∩*)

(*∩*)

converts S to a graph with slices w^ *|*E*i*. So, *{} ≈* ^x, w^ *≈ { }* and w^ *|*E *≈ {*w^ *|*E*}*.

Structural rules transform slices and graphs. The *singleton rules* convert a slice to its singleton graph and vice-versa. The *promotion rule* converts an expression E to the slice x*|*E. *Rule* (*→V*) converts slice P + w*|*G to the graph *{*P+ w*|*S */* S *∈* G*}*, and *rule* (*→Λ*) converts slice P+w*|*S to the glued slice PwS. We thus have a derived

^

rule (*→*) converting slice P + w*|*G to the glued graph PwG. So, P + w

*|{ }*

(*→*)

*≈ { }*.

The *zero rule* (*Z*) erases a zero slice. The *alternative expansion rule* (w *|* E) expands a slice S to graph *{* S+ w*|*E*,* S+ w*|*E *}*. From (w *|* E) and (*Z*), we can derive

the *shift* expansion rule: a slice P with 1-ary arc u ,˛ w^ L /w*¸j*  ,˛E

and 2-ary arc u L u*j*, expands to P + u*j|*E. So, a slice with slice witness

*−→*

w L /v*¸* ,˛E shifts to a slice with expression witness E. [7](#_bookmark12)

^

The *expression-set* rule converts set *{* E1*,...,* E*n }* to the slice E1¸ Tables [1](#_bookmark13) and [2](#_bookmark14) summarize these conversion rules.

*...*

^x

¸,E*n* .

7 This shift expansion rule can be used to simulate the modal [ ] transfer [[7](#_bookmark42)].

(SG)

P

Table 1

General graph calculus conversion rules (cf. p. [5](#_bookmark8))

(GS)

(*†*)

,˛

*≈ {* P *} {* P *} ≈* P E *≈* ^x E

P+ w ,˛G (*→∨*) *{*P+ w ,˛S */* S *∈* G*}* P+ w ,˛S (*→∧*)

*≈*

*≈*

PwS

⎧⎪ S1 ⎫⎪

x

⎧⎪ E1 rz w^ ⎫⎪

⎨ . ⎬

⎪ ⎪

(*∪*)

*≈*

˛

^

w

˛ ˛¸

˛¸

^

(*∩*)

*≈* .

⎨

⎬

⎪ ⎪

⎪⎪⎩

⎪⎪⎭

S1 r

*...* S*n*

E1 r

*...* E*n*

⎪⎪⎩ ˛

⎪⎪⎭

S*n*

 ()

E *≈* E *{* E1*,...,* E*n }*

(**E**)

*≈*

E1 .

w^

*...*

,E*n*

¸,E*n*

⎧ ^x ⎫

(*2*)

S

S *≈ { }* (S with conflict)

(w*|*E)

*≈*

⎪⎨ S + w ,˛E *,* ⎪⎬

Table 2

Derived graph calculus rules

⎪⎩ S + w ,˛E ⎪⎭

P+ w ,˛G (*→*)

*≈*

PwG

P+ u ,˛

L /*¸* ˛

(*~*)

P+ u ,˛

L /*¸* ˛

w w*j*

^

L

*j*

,E *≈*

L

*j*

w^ w*j* ,E

,˛

u u E

We also have rules for capturing special properties of a relation.

**(**Rf[L]**)** For a reflexive relation L: expand node w to w ,*¸*L .

L z*˛* L z*˛*

L z*˛* L z*˛*

**(**Tr[L]**)** For a transitive relation L: expand u v

1. to u

v w.

L

L z*˛* L z*˛*

**(**Sm[L]**)** For a symmmetric relation L: expand u

v to u ,*¸* v .

L

**(**As[L]**)** For an anti-symmmetric relation L: identify nodes u and v with u ¸*,*L /v*¸*.

¸L*,* /*¸*

L z*˛*

We often use u

v as short for u ¸*¸* v .

L

The *general graph calculus* consists of the conversion rules in Table [1](#_bookmark13) (p. [7](#_bookmark13)). A *graph calculus* extends the general one by some rules for properties of relations (as above). A *derivation* is a finite sequence of rule applications. The general calculus is sound for equivalence: if E derives F, then E *≡* F.

**Proposition 3.2 (Graph calculi)** *Each graph calculus is refutationally sound and complete: a ﬁnite expression set* E *is null iff* E *derives the empty graph { }. b*

**Proof.** Soundness is clear: the rules involve equivalent expressions. For complete- ness: if E does not derives *{ }*, then we can obtain a chain of non-zero slices, whose underlying drafts have as co-limit a non-zero sketch Σ; E is non-null in the natural structure N[Σ]. If a rule like (Tr[L]) is present, then N[Σ] will be L-transitive, as Σ is “saturated”. For more details, see [[14](#_bookmark45),[15](#_bookmark50)]. *2*

# Intuitionistic Modal Logic: Flat Semantics

We now examine flat semantics for intuitionistic modal logic, akin to that in [[13](#_bookmark47)]. It is convenient to consider a hierarchy of structures.

A *pre-relational structure* B consists of a set W */*= *∅* (of worlds) with a special binary relation *≤* on W, together with a binary relation rB on W (for r *∈* RS) and a valuation Vl : PL *→℘*(W). We use the abbreviations P for pB := Vl(p) and R for rB. We also introduce function Lv : W *→℘*(PL) by u *∈* W *'→ {*p *∈* PL */* u *∈* Vl(p)*}*.

Formula *satisfaction* (with H as short for HB) is as follows. The local cases are: u */*H *⊥*; u H p iff u *∈* Vl(p) (i. e., p *∈* Lv(u)); u H *ψ ∧ θ* iff u H *ψ* and u H *θ*; u H *ψ ∨ θ* iff u H *ψ* or u H *θ*. For *⟨ ⟩*: u H *⟨*r*⟩ϕ* iff, for some v *∈* W, uRv and v H *ϕ*. For *→*: u H *ψ → θ* iff, for every v *≥* u, if v H *ψ*, then v H *θ* (i. e., there exists no v *≥* u such that v H *ψ* and v */*H *θ*). For [ ]: u H [r]*ϕ* iff, for all v*,* w *∈* W, if v *≥* u and vR w, then w H *ϕ* (i. e., there exist no v*,* w *∈* W such that v *≥* u, vR w and w */*H *ϕ*). Thus, for

*¬*: u H *¬ϕ* iff, for every v *≥* u, v */*H *ϕ* (i. e., there exists no v *≥* u such that v H *ϕ*).

A *relational structure* is a pre-relational structure B where relation *≤* is a partial order on W. To have monotonicity of satisfaction, one restricts relational structures to *birelational structures* by imposing 3 extra requirements. Monotone valuation: if u *≤* u*j*, then Lv(u) *⊆* Lv(u*j*). (F1): given u*j,* u*,* v *∈* W, such that u*j ≥* u and uR v, there exists v*j ∈* W, such that u*j* R v*j* and v *≤* v*j*. (F2): given u*,* v*,* v*j ∈* W, such that uRv and v *≤* v*j*, there exists u*j ∈* W, such that u *≤* u*j* and u*j* R v*j*. [[13](#_bookmark47), p. 50]

To reason graphically about flat semantics with a symbol wc for *≤*, we consider a graph language GLf with Sb1 := Φ and Sb2 := RS*∪{*wc*}*. We draw wc-arrows as /. *¸* A pre-relational structure B gives a structure for GLf with *ϕ*B := *{*u *∈* W */* u HB *ϕ}*.

Then, we can handle logical symbols by the 6 *pre-relational elimination rules*

converting formulas to equivalent expressions given in Table [3](#_bookmark17).

The pre-relational elimination rules in Table [3](#_bookmark17) transcribe formula satisfaction in graph terms, which guarantees their soundness. For instance, for (*⟨ ⟩*), we have: s *∈* (*⟨*r*⟩ϕ*)B iff, for some v *∈* W, uRv and v HB *ϕ*, i. e., assignment g with xg = u

and yg

= v satisfies draft x

r z*˛* ˛

y ,*ϕ* iff u *∈* [ ^x

r

 y ,*ϕ* ]B. Also,

z*˛* ˛

for (*→*), we have: u */∈* (*ψ → θ*)B iff s */*HB *ψ → θ* iff there is some v *≥* u such that

v HB *ψ* and v */*HB *θ*, i. e., assignment g with xg = u and yg = v satisfies draft

Table 3

Pre-relational elimination rules

Formula *≈* Expression Comment

(*⊥*)

*⊥ ≈ { }* empty graph

(*Λ*)

z ,˛

*ψ ∧ θ*

*≈ ψ* r

^x

*θ* single-node slice

(*V*)

*ψ ∨ θ ≈*

⎧⎨ *ψ* rz

^x *,* ⎫⎬

graph with single-node slices

⎩ ^x ,˛*θ* ⎭

*⟨*r*⟩ ϕ*

z*˛* ˛

(*⟨ ⟩*)

*≈* ^x

(*→*)

*≈*

r

y ,*ϕ* 2-node slice

¸,*ψ*

*ψ → θ*

^x

([ ])

complemented 2-node slice

r z*˛* ˛

y*¸*

,¸*θ*

[r] *ϕ*

*≈* ^x  y

z ,*ϕ* complemented 3-node slice

1.  y

˛,*ψ*

,¸*θ*

iff u is in the extension of slice

^x  y

¸,*ψ*

,¸*θ*

in B.

One can also consider some variations. For the condition “w H *⟨*r*⟩ϕ* iff, there

are w0*,* v0 *∈* W such that w *≥* w0, w0 R v0 and v0 H *ϕ*” (attributed to Plotkin

z ,*ϕ*

and Stirling [[13](#_bookmark47), p. 49]), we obtain the slice

^x ¸*¸* y

r z*˛* ˛ . We could

similarly handle a condition like “w H *⟨*r*⟩ϕ* iff, for all w*j ∈* W, if w*j ≥* w there exists

v*j ∈* W, such that w*j* R v*j* and v*j* H *ϕ*”.

The next result illustrates how one can obtain expressions for complex formulas from those of its immediate sub-formulas.

**Proposition 4.1 (Derived pre-relational conversions)** *The pre-relational conversions in Tables* [*4*](#_bookmark19) *and* [*5*](#_bookmark21) *are derived. b*

**Proof.** By graph rules and pre-relational elimination rules. For instance, (*¬*) is

([ ])

r z*˛* ˛ ()

r z*˛* ˛

clear; for ([ ]): [r] *ϕ ≈* ^x yz  ,*ϕ*

*≈* ^x  y

z ,*ϕ* ; for

(*ч*)

(*¬ ¬*): *¬¬ ϕ*

*≈*

˛ ( ;*ч*)

^x  y ,˛ ^x

y,˛*ϕ* .

^x y

,*¬ϕ*

*≈*

([ ])

r z*˛* ˛

(*Λ*)

For ([ ] *∧*): [r] (*ψ ∧ θ*)

*≈* ^x  y

z ,*ψ ∧ θ ≈*

r z*˛* ˛

˛,*ψ*

(*∩*;*→*)

⎧⎪

⎪⎨ ^x

⎪ z*˛*

r z*˛*

y z

¸,*ψ* ⎫⎪

⎪⎬

⎪

(*∪*)

^x yz

,

^ ,¸*θ*

z

*ψ*  z r*.* y

rz

¸*¸* ^x

x

*≈*

,˛

*≈*

r

⎪⎪⎩ ^x /y*¸* z

,¸*θ* ⎪⎪⎭

r z*˛* ˛

r ^x

^x y

z ,*θ* . *2*

Table 4

Derived pre-relational slice conversions (the nodes with ‘\*’ are new)

(*⊥*) S + w ,˛*⊥ ≈∗ {}* S + w ,˛*⊥ ≈∗* S + w

(*Λ*) S+ w ,*ψ ∧ θ ≈∗*

˛

S+ w ,*ψ*

˛

S+ w ,˛*ψ ∧ θ ≈*

*∗* ⎧⎨ S+ w ,˛*ψ ,* ⎫⎬

˛¸*θ*

⎩ S+ w ,˛*θ* ⎭

˛

(*V*) S+ w ,*ψ ∨ θ ≈*

*∗* ⎧⎨ S+ w ,˛*ψ ,* ⎫⎬

S+ w ,*ψ ∨ θ ≈∗*

˛



S+ w ,*ψ*

˛ *θ*

˛

⎩ S+ w ,˛*θ* ⎭

( ) S + u ,˛*¬ϕ ≈∗* S + u

ר*z*

S + u ,˛*ψ → θ ≈∗* S + u *ψ*

¸

ר*z*

,

˛

v*∗*  ,*ϕ*

v*∗*  ,˛*θ*

(*⟨ ⟩*) S+ u ,˛*⟨*r*⟩ϕ ≈∗* S+ u

r

S+ u ,˛*⟨*r*⟩*(*ψ ∧ θ*) *≈∗* S+ u *ψ*

¸

r

,

v*∗* ,˛*ϕ*

v*∗* ,˛*θ*

**Example 4.2** (Pre-relational consequence) To show that [r] *ψ* is a consequence of

[r] (*ψ ∧ θ*), we first convert [r] (*ψ ∧ θ*) *≈∗* S*j* and [r] *ψ ≈∗* S*jj* (cf. Table [5](#_bookmark21)). We have:

*j*

[r] (*ψ ∧ θ*)

,

r

S,r

(*→∧*)2

^x

˛z

*ψ*,$

z,*7* y ,*¸* ^x

r

rz

^x

zy

,˛ ^x

r

¸y

˛,*θ*

zz

S

*∗*

[r] *ψ*

*≈* ^x *≈*

˛*j*z*j*

The resulting slice S is zero: slice S has as witness at node x the slice

r

z,*7*

*ψ*,"

r

*ψ* ,\ z,*7* y ,*¸*^x under morphism x *'→* x, y *'→* y, z *'→* z. (Notice that S can be

Table 5

Derived pre-relational expression conversions

(*ч*) *¬ ϕ ≈∗* ^x  y

^x

*¸*

y

,˛

^x

*¸*

y

,

˛*ϕ*

(*ч ч*) *¬¬ ϕ ≈∗*

,˛*ϕ* complemented 2-node slice

*¬¬ϕ ≈* ^x

y*¸* ,˛ ^x

y*¸* ,˛*ϕ*

(*ч⟨ ⟩*) *¬ ⟨*r*⟩ ϕ ≈∗* ^x

(*ч ч*) *∗*

1. r zz *˛* ˛*ϕ*

,

r z*˛* ˛ ˛

(*⟨⟩ ч*) *⟨*r*⟩¬ ϕ ≈∗* ^x

y , ^x y,*ϕ* z

([ ]) [r]*ϕ ≈∗* ^x  y

([ ] *ч*) [r] *¬ ϕ ≈∗*

(*ч* [ ]) *¬* [r] *ϕ ≈∗*

r

z ,*ϕ*

r

^x

^x

y

z*˛*

z

w*¸*

˛

,*ϕ*

y

,

˛

^x

y

r

z*˛*

z

,˛*ϕ*

z*˛* ˛

˛ r

r

*ψ* rz zy ,*¸* ^x

,

([ ] *Λ*) [r] (*ψ ∧ θ*) *≈∗*

rz ^x

^x ¸y

zz

,˛*θ*

shifted to a slice with expression witness *ψ* at z.) Thus S is unsatisfiable. Hence, set *{*[r] (*ψ ∧ θ*)*,* [r] *ψ}* cannot be satisfied in a pre-relational structure. *q*

We now indicate how our calculus handles the necessitation rule, namely: “from theorem *ψ*, infer theorem [r] *ψ*”. Its refutational analogue can be formulated as: “if *ϕ ≈∗ { }*, then [r] *ϕ ≈∗ { }*”. Now, if *ϕ ≈∗ { }*, then we will have the conversions:

([ ])

r z*˛* ˛ *∗*

r z*˛* ˛

(*→*)

[r] *ϕ ≈* ^x  yz ,*ϕ ≈* ^x  yz ,*{} ≈ { }*.

We also have rules coming from the intended meaning of wc as *≤*. The *relational operational rules* are (Rf[wc]), (As[wc]) and (Tr[wc]) (cf. Section [3](#_bookmark7), p. [7](#_bookmark14)).

A sketch, or slice, of GLf is wc*-reduced* iff u = v, whenever it has arcs u ¸*,* /v*¸*.

Every GLf-slice can be contracted to a wc-reduced slice. [8](#_bookmark23)

For birelational structures, we also have the 3 *birelational transformation rules* :

*¸*

(p) Contract slice S + p rz u

u*j¸* ,˛p to the empty graph *{ }*.

r v*˛* ˛

r

 *z* ˛

8 For instance, ^x ¸*¸* y,p is not wc-reduced, but it contracts to the wc-reduced ¸^z ,*,¸* ,p.

*j* r z*˛ j* r z*˛ ∗*

(F1) Expand slice S + u ¸*¸* u

v to S + u ¸*¸* u

v (with new v ).

r z*˛ j*

r v*∗* ¸*"*

r z*˛ j ∗*

(F2) Expand slice S + u

v  v*¸*to S + u

v v¸*¸,*(with new u ).

zu*#∗#* r

Then, we can derive the following *birelational formula transfer* conversion:

S+ *ϕ* rz u  u*j¸≈∗* S + *ϕ* rz u  u*j¸* ,˛*ϕ* . (By the alternative ex-

*¸*

*¸*

pansion rule (w *|* E): case *⟨*r*⟩ ϕ* follows from (F1) and case [r] *ϕ* follows from (Tr[wc])).

**Example 4.3** (Birelational consequence) To show that *¬¬ϕ* is a birelational con- sequence of *ϕ*, we consider the set *{ϕ, ¬¬ ϕ}* and use (*¬ ¬*) (cf. Table [5](#_bookmark21), p. [11](#_bookmark21)).

1. *{ϕ, ¬¬ϕ} ≈∗* S1, with S1 = *ϕ* rz ^x

z

^x y*¸*

,˛*ϕ* .

1. By reflexivity (Rf[wc]), slice S1 expands to the following slice S2:

,˛

*ϕ* rz ^x

z,*¸*

^x y*¸*

,˛*ϕ*

1. Transfer formula *ϕ* from x to z to obtain the following slice S3:



,˛

,˛

*ϕ* rz ^x



*ϕ* r˛

z,*¸*

^x y*¸*

,˛*ϕ*

Slice S3 is zero: slice S3 has as witness at node z the slice x /y*¸* ,˛*ϕ* under morphism x*,* y *'→* z. (Notice that S3 can be shifted (cf. p. [6](#_bookmark11)) to a slice with expression witness *ϕ* at z.) So, S3 is unsatisfiable.

^

Hence, set *{ϕ, ¬¬ϕ}* cannot be satisfied in a birelational structure. *q*

We can show graphically that the following formulas (cf. [[13](#_bookmark47), p. 51, 52]) are birelationally valid: [r](*ψ → θ*) *→* ([r]*ψ →* [r]*θ*), [r](*ψ → θ*) *→* (*⟨*r*⟩ψ → ⟨*r*⟩θ*),

*¬⟨*r*⟩⊥*, *⟨*r*⟩*(*ψ ∨ θ*) *→* (*⟨*r*⟩ψ ∨ ⟨*r*⟩θ*), (*⟨*r*⟩ψ →* [r]*θ*) *→* [r](*ψ → θ*) and *¬⟨*r*⟩ϕ →* [r]*¬ϕ*. (In fact, *¬⟨*r*⟩⊥* can be seen to be pre-relationally valid.)

The natural construction (cf. Section [3](#_bookmark7), p. [5](#_bookmark8)) applied to a proper wc-reduced

GLf-sketch Σ gives a pre-relational structure B[Σ].

**Example 4.4** (Birelational non-consequence) To show that p is not a birelational consequence of *¬¬*p, we consider the set *{¬¬*p*,* p*}* and use (*¬ ¬*) (cf. Table [5](#_bookmark21), p. [11](#_bookmark21)).

1. Set *{¬¬*p*,* p*}* converts to a slice, which expands by (Rf[wc]) to slice S1:

z ^ ,˛ ˛ ˛

pr x

^x y*¸* , ^x

y*¸* ,p

1. Now, slice S1 shifts (cf. Section [3](#_bookmark7), p. [6](#_bookmark11)) to the following slice S2:



z ˛

^ , ˛ ˛

pr x

˛z

p rz y ¸*,* ^x

^x y*¸* , ^x

y*¸* ,p

1. By (), lowering (*→Λ*) (cf. Table [1](#_bookmark13)) and reflexivity (Rf[wc]) , we have slice S3:



z ˛

^ , ˛ ˛

x

pr

p rz

^x

y , *7*

y*¸* , ^x

y*¸* ,p

1. The positive part of S3 and corresponding natural structure B are:

x/y*¸* *¸* ,˛p

*≤ ≤*

,˛P



x /y *¸*

*<*

Note that structure B is birelational. Much as in Example [3.1](#_bookmark10) (Natural con- struction), we see that the identity assignment 1 satisfies draft S3 in B.

Hence, these slices and set *{¬¬*p*,* p*}* are satisfiable in a birelational structure. Thus, p is not a birelational consequence of *¬¬*p. *q*

The special binary relation *≤* of a pre-relational structure may be symmetric.

For such cases, we use the rule (Sm[wc]) (cf. Section [3](#_bookmark7), p, [7](#_bookmark14)).

**Example 4.5** (Symmetric birelational consequence) To show that *ϕ* is a symmetric birelational consequence of *¬¬ϕ*, we consider the set *{¬¬ϕ, ϕ}* and use (*¬ ¬*).

1. Set *{¬¬ϕ, ϕ}* converts to the following slice S1:

*ϕ* rz ^x ,˛ ^x

zy*˛* ,˛

^x

zy*˛* ,˛*ϕ*

1. By (Rf[wc]), graph rules and symmetry (Sm[wc]), we transform S1 to slice S2:

*ϕ* rz

&*j*

^x ,*¸*

,˛ ^x

zy*˛* ,˛

^x

zy*˛* ,˛*ϕ*

v*z* ,˛

y *ϕ*

1. Now, transfer formula *ϕ* from y to x (cf. p. [12](#_bookmark22)), expanding S2 to the slice S3:

*ϕ*¸(



*ϕ* rz

&*j*

^x ,*¸*

,˛ ^x

zy*˛* ,˛

^x

zy*˛* ,˛*ϕ*

vy *z* ,˛*ϕ*

This slice S3 has a conflict at node x (with formula *ϕ* as expression witness). So, in a birelational structure with symmetric *≤*, one cannot satisfy *{¬ ¬ ϕ, ϕ}*. *q*

We can similarly show that *ϕ∨ ¬ϕ* is valid in symmetric birelational structures. The flat graph calculi are as follows. The *pre-relational calculus* is the graph calculus for graph language GLf with the elimination rules in Table [3](#_bookmark17). The *relational calculus* is the extension of the pre-relational calculus by the rules (Rf[wc]), (As[wc]) and (Tr[wc]) (cf. p. [11](#_bookmark20)). The *birelational calculus* is the extension of the relational calculus by the rules (p), (*F* 1) and (*F* 2) (cf. p. [11](#_bookmark22)). The *symmetric flat calculus* is the extension of the birelational calculus by the symmetric rule (Sm[wc]) (cf. p. [13](#_bookmark26)).

**Theorem 4.6 (Flat calculi)** *The flat graph calculi are sound and complete. b*

**Proof.** By Proposition [3.2](#_bookmark15): Graph calculi (p. [8](#_bookmark15)). *2*

# 5 Intuitionistic Modal Logic: Decoupled Semantics

We now examine another semantics for intuitionistic modal logic.

The motivation comes from decoupling objects and stages, as in [[5](#_bookmark41)]. A *stratiﬁed structure* consists of a set *I* (of stages) partially ordered by *≤* with, for each *i ∈ I* : a universe *Ci /*= *∅* (of objects), a subset p C *⊆ Ci* (for p *∈* PL) and binary relation r C on *Ci* (for r *∈* RS). We use the abbreviations: P*i* for p C and R*i* for r C .

*i*

*i i* *i*

We prefer another formulation as follows. A *pre-graded structure* C consists of

2 sets *I* (of stages), with a special binary relation *≤* on it, and *C /*= *∅*; it has as domain a non-empty subset *C×* of *C × I* and (with abbreviations P for pC and R for rC) a subset P *⊆ C×* (for p *∈* PL) and 2-ary relation R on *C×* (for r *∈* RS) such that *i* = *j* whenever *⟨a, i⟩* R *⟨b, j ⟩*. We can introduce a special relation on ordered pairs by *⟨a, i⟩ ≤ ⟨b, j ⟩* iff *i ≤ j* . We then obtain a pre-relational structure.

*Satisfaction* (with H as short for HC) is as follows. For *⊥*, p, *∧*, *∨* and *⟨ ⟩*, it is as in Section [4](#_bookmark16), with u = *⟨a, i⟩*. For *→*: *⟨a, i⟩* H *ψ → θ* iff , for every *j ≥ i* , if *⟨a, j ⟩* H *ψ* then *⟨a, j ⟩* H *θ* (i. e., there exists no *j ≥ i* such that *⟨a, j ⟩* H *ψ* and *⟨a, j ⟩ /*H *θ*). For [ ]: *⟨a, i⟩* H [r]*ϕ* iff, for all *j ≥ i* and *b ∈ C*, if *⟨a, j ⟩* R *⟨b, j ⟩* then *⟨b, j ⟩* H *ϕ* (i. e., there exist no *j ≥ i* and *b* such that *⟨a, j ⟩* R *⟨b, j ⟩* and *⟨b, j ⟩ /*H *ϕ*). Thus, for *¬*: *⟨a, i⟩* H *¬ϕ* iff, for every *j ≥ i* , *⟨a, j ⟩ /*H *ϕ* (i. e., there exists no *j ≥ i* such that *⟨a, j ⟩* H *ϕ*).

As in Section [4](#_bookmark16) (p. [8](#_bookmark16)), we consider some restrictions. A *graded structure* is a pre-graded structure C where special relation *≤* is a partial order on *I* . A *growing- graded* structure is a graded one with growing universes, predicates and relations. For *i ≤ j ∈ I* : if *⟨a, i⟩ ∈ C×* then *⟨a, j ⟩ ∈ C×* (i. e., *Ci ⊆ Cj* ); if *⟨a, i⟩ ∈* P then

*⟨a, j ⟩ ∈* P (i. e., P*i ⊆* P*j* ); if *⟨a, i⟩* R *⟨b, i⟩* then *⟨a, j ⟩* R *⟨b, j ⟩* (i. e., R*i ⊆* R*j* ). [9](#_bookmark31) On a growing-graded structure, satisfaction can be seen to be monotonic.

We wish to reason graphically about decoupled semantics with symbols ic (for *≤*) and eo (with intended meaning *⟨a, i⟩*eo*⟨b, j ⟩* iff *a* = *b*). For this purpose, we consider a graph language GLd with Sb1 := Φ and Sb2 := RS *∪ {*ic*,* eo*}*. We draw ic-arrows as

/*¸*a*¸*nd eo-arrows as /. *¸*A pre-graded structure C gives a structure for GLd with *ϕ*C := *{⟨a, i⟩∈ C× / ⟨a, i⟩* HC *ϕ}*.

Then, we can handle logical symbols by 6 *pre-graded elimination rules* converting formulas to equivalent expressions, much as before. The rules for *⊥*, p, *∧*, *∨* and

*⟨⟩* are as in Table [3](#_bookmark17) (p. [9](#_bookmark17)). The other 2 rules convert formulas *ψ → θ* and [r] *ϕ*,

respectively, to the expressions

^x )y*,*

¸,*ψ*

,¸*θ*

z

and zr

)*,*

^x  y

zz *˛* ,˛*ϕ* .

Also, formula *¬ ϕ* converts to the expression Thus, we have derived pre-graded conver

^

x

)y*,*

z˛

,*ϕ*

sions much as those

.

in Proposition [4.1](#_bookmark18).

Also, the intended meanings of eo and ic lead to some operational rules as follows

(cf. Section [3](#_bookmark7), p. [7](#_bookmark14)). For eo, we have (Rf[eo]), (Sm[eo]) and (Tr[eo]). For graded ic, we have (Rf[ic]), (As[ic]) and (Tr[ic]), as well as the rule identifying nodes u and v such

that u ¸ *¸.* zv *˛*. For a symmetric *≤*, we use (Sm[ic]).

¸*¸*

For growing-graded structures, we also have the following 3 *growing transfor-*

*mation rules*. For domain: given u  v, add u zu*∗\_ \_* v(with new

¸ *\_*

node u*∗*). For p *∈* PL: erase slice with p rz u zv*˛* ,˛p . For r *∈* RS: given

,*c* r z*˛*

u*j*

*¸*

u

v

¸*¸*

zv*j˛*, add u*j*

v*j¸*. Then, we can derive the growing

r

formula transfer: S + *ϕ* rz u

zv*˛≈∗* S + *ϕ* rz u

zv*˛* ,˛*ϕ* .

We can establish consequence as in Section [4](#_bookmark16), with ic and eo in lieu of wc. We can show that [r] *ψ* is a pre-graded consequence of [r] (*ψ∧θ*) as in Example [4.2](#_bookmark20), that *¬¬ϕ* is a growing-graded consequence of *ϕ* as in Example [4.3](#_bookmark24), and that *ϕ* is a symmet- ric growing-graded consequence of *¬¬ϕ* as in Example [4.5](#_bookmark27) (notice that symmetric growing-graded structures have constant universes, predicates and relations).

We can also establish non-consequence much as in Section [4](#_bookmark16), even though the natural construction is now more involved. A sketch is *∩-reduced* iff u = v, whenever

it has arcs u ¸\_¸*\_c\_* zv*¸\_*. The natural construction applied to a proper *∩*-reduced GLd-

sketch Σ = *⟨*N; A*⟩* gives a pre-graded structure C[Σ] and assignment hΣ as follows.

1. Define 2-ary relations on N: u *∼*eo v iff uzv *∈* A and u *∼*ic

v iff u ,*¸* v *∈* A.

We have equivalences, with quotient sets *I* := N*/* ic

*~*

and *C* := N*/*eo .

1. Define domain *C×* := *{⟨*[w]eo*,* [w]ic*⟩∈ C × I /* w *∈* N*}* and special relation *≤* on

*~*

*I* by [u]ic *≤* [v]ic iff u  v *∈* A.

9 Notice that these restrictions are simpler and more intuitive than requirements (F1) and (F2) in Section [4](#_bookmark16).

1. Define subsets by *⟨*[w]eo*,* [w]ic*⟩ ∈* P iff w ,˛p *∈* A and relations by

r z*˛*

*⟨*[u]eo*,* [v]ic*⟩* R *⟨*[v]eo*,* [v]ic*⟩* iff u v *∈* A.

1. Define natural assignment hΣ :N *→ C×* by w *'→ ⟨*[w]eo*,* [w]ic*⟩*.

To see that p is not a growing-graded consequence of *¬¬*p, we proceed as in Example [4.4](#_bookmark25) (Birelational non-consequence, p. [12](#_bookmark25)), with ic and eo in lieu of wc, as well as (Rf[ic]), (Rf[eo]) and (Sm[eo]). We obtain the following final slice S*j* :

3

p rz

p rz

y1 ¸*1 c*

t*\*

˛^x*0*,*¸* ,



¸*,*,*¸*

˛

z*5*

^x )*,*

y ,˛

v*˛*

^x

y ,˛p

The positive part D of S*j*

3

and corresponding natural structure C are:

*≤*



t*\*

p rz

7 *7*

˛^x*6*,*¸*

¸y *,*¸*"*

,*¸*

8*r*

[x]eo=[y]eo

*≤ ⟨*[x]eo*,* [x]ic*⟩*

8*r*

*>*

*c*

ic

eo

ic

[x] *≺*[y]

ic

P rz

*⟨*[y]

*,* [y] *⟩*

Notice that structure C is growing-graded.

We can see that the natural assignment h satisfies draft D in structure C:



zx *z*

,*¸*

D p rz y ,¸ *c,*

¸*,*,*¸*

h

C Prz

z*#*

*⟨*[y] *,* [y] *⟩*

eo ic

,¸*,¸*

h

*≤*

8,*r7*

*≤ > ⟨*[x]eo*,* [x]ic*⟩*

Much as before, we can see that the natural assignment h satisfies draft S*j* in

3

structure C. Thus, these slices and set *{¬¬*p*,* p*}* are satisfiable in a growing-graded structure. Hence, p is not a growing-graded consequence of *¬¬*p.

The *decoupled graph calculi* are as follows (cf. p. [15](#_bookmark30)). The *pre-graded calculus* is the graph calculus for graph language GLd with the 6 pre-graded elimination rules and the 3 equivalence rules for eo. The *graded calculus* is the extension of the pre- graded calculus by the rules (Rf[ic]), (As[ic]), (Tr[ic]) and the rule identifying nodes u

and v with u ¸ *¸.* zv *˛*. The *growing calculus* is the extension of the graded calculus

¸*¸*

by the growing rules. The *symmetric decoupled calculus* is the extension of the growing calculus by the symmetric rule (Sm[ic]).

**Theorem 5.1 (**GLd **calculi)** *The decoupled calculi are sound and complete. b*

**Proof.** By Proposition [3.2](#_bookmark15): Graph calculi (p. [8](#_bookmark15)). *2*

# Extension to Multi-modal Logics

We have sound and complete graph calculi for flat and decoupled intuitionistic modal logics. We now indicate how to extend these calculi to multi-modal logics.

We can also allow some connections as well as some operations on relations (much as in [[3](#_bookmark39)]). For instance, we can express inclusion of relations by a rule

*−→*

(L *±* K) adding u K

*−→*

v whenever we have u L

v and intersection of relations by

a rule (L*H*K) adding u L zv*˛*whenever we have u L *H* K /v*¸*. We can similarly

K

express composition (by consecutive arrows), transposal (by arrow reversal) and

identity (by node identification) [[14](#_bookmark45),[15](#_bookmark50)]. For a set Δ of constraints, a Δ*-derivation* is a finite sequence of rule applications and constraints in set Δ.

Consider relation symbols r, s*j*, s*jj* and t, subject to the restrictions: “r *⊆* s*j ∩* s*jj*, s*j ⊆* t and t is transitive”. We construct a graph calculus by adding to our basic rules the set Δ consisting of the rules (r *±* s*j H* s*jj*), (s*j ±* t), (s*j H* s*jj*) and (Tr[t]).

(+) We can show that *⟨*t*⟩ ϕ* is a Δ-consequence of *⟨*r*⟩⟨*s*j⟩ ϕ*, much as before: we trans- form the set *{ ⟨*r*⟩⟨*s*j⟩ ϕ, ⟨*t*⟩ ϕ}* to the following slice:

r z v¸,*> ,, j ϕ*

*j*

,

s

¸

s

s*jj*

*ϕ*rz y ¸*,*t ^x

rz

u^ t

t

t zj*˛z*

)*,*

w

(*—*) We can also obtain a Δ-model for *{ ⟨*r*⟩* p*, ⟨*t*⟩⟨*t*⟩* p *}*, much as before. We transform this set to a slice S, which gives a model N, as follows:

s*j*

, *c*r

S*j*

˛

y ,R

,¸*¸¸*

*c*

x

p rz

y ,¸*¸¸* ^x

s*jj*

t

, ^u t /v*¸* t /w*¸*

,˛p P rz

T

S*jj*

Now, consider graph languages GLf (cf. Section [4](#_bookmark16)) and GLd (cf. Section [5](#_bookmark28)).

**Lemma 6.1 (Equivalent calculi)** *A modal formula is flat (relational or birela-* *tional) derivable iff it is decoupled (graded or growing) derivable. b*

**Proof.** We can transform derivations back and forth. [10](#_bookmark35) *2*

**Theorem 6.2 (Equivalent semantics)** *The same modal formulas hold in flat (relational or birelational) and decoupled (graded or growing) structures. b*

10 Call expressions E of GLf and F of GLd are associated (EF) iff F is the result of replacing everywhere in E wc by ic and eo, and similarly for constraints. For sets of constraints Δf in GLf and Δd in GLd, ΔfΔd iff each *δ ∈* Δf has some associated *δ′ ∈* Δd and vice-versa. Call derivations E1*,...,* E*n* and F1*,...,* F*n* associated iff E*i* F*i*, for *i* = 1*,..., n*. Call an expression of GLd neat iff ic and eo occur only in parallel arcs, and similarly for (sets of) constraints and derivations. By *≤ '→* ic*,* eo we transform flat rules to decoupled ones and vice-versa. So, given associated constraint sets ΔfΔd, every flat Δf-derivation Πf has an associated neat decoupled Δd-derivation Πd (which will be graded or growing whenever Πf is relational or

birelational) and, similarly, every neat decoupled Δd-derivation Πd has an associated flat Δf-derivation Πf.

**Proof.** By Lemma [6.1](#_bookmark34) and completeness: Theorems [4.6](#_bookmark29), p. [14](#_bookmark29), and [5.1](#_bookmark32), p. [16](#_bookmark32). *2*

# Concluding Remarks

We have presented a flexible and uniform formalism for intuitionistic modal logics where one can express, analyse and compare possible-world semantics. Our ap- proach explores the flexibility of graph caluli [[14](#_bookmark45),[15](#_bookmark50)] to express directly and graph- ically Kripke-based semantics of intuitionistic modal logics.

We have illustrated these ideas by applying them to two semantics (in Sec- tions [4](#_bookmark16) and [5](#_bookmark28)) and indicated their extension to multi-modal logics in Section [6](#_bookmark33). Our approach is uniform: once we have expressed the semantics (including connections among relations), we apply the corresponding (sound and complete) graph-calculus. For flat and decoupled semantics, we have transcribed their satisfaction conditions graphically to expressions and used this to show that they give equivalent semantics (in Section [6](#_bookmark33)). We have also illustrated (in Section [4](#_bookmark16)) how one can express simple variations of the satisfaction conditions, which give different semantics on relational structures, though some of them may coincide on birelational structures.

We would like to stress some distinctions between graph calculi and other meth- ods for handling logics. Natural deduction relies on rules for introducing and eliminating logical operators (connectives, etc.) and its aim is building derivation trees [[13](#_bookmark47)]. Sequent calculi uses rules for left and right introduction of logical opera- tors and its aim is building sequent trees [[8](#_bookmark43),[9](#_bookmark44),[10](#_bookmark48),[12](#_bookmark49)]. In tableaux, the emphasis is on rules that describe truth/falsity conditions for logical operators and the aim is con- structing refutation trees [[1](#_bookmark36),[7](#_bookmark42)]. Graph calculi employ graphical interpretations of logical operators and the aim is building graphical objects that represent conditions on models; their visual features render them attractive to human users.

We thus have a flexible, uniform, rigourous and intuitive formalism for visual exploration of intuitionistic multi-modal logics.

# References

1. Amati, G., and F. Pirri, *A uniform tableau method for intuitionistic modal logics I*, Studia Logica **53**: 1 (1994), 29–60.
2. Blackburn, P., M. de Rijke and Y. Venema, “Modal Logic”, Cambridge U. Press, Cambridge, 2001.
3. Catach, R., Normal modal logics, In: Proc. AAAI-88 (1988), 491–495.
4. Curtis, S., and G. Lowe, *Proofs with graphs*, Science of Computer Programming **26** (1996), 197–216.
5. Ewald, W. B., *Intuitionistic tense and modal logic*, J. of Symbolic Logic **51** (1986), 166–179.
6. Fischer Servi, G., Semantics for a class of intuitionistic modal calculi, In: dalla Chiara, M. L. (ed.) Italian Studies in the Philosophy of Science, 59–72, D. Reidel, Dordrecht, 1981.
7. Fitting, M., “Proof Methods for Modal and Intuitionistic Logics”, D. Reidel, Dordrecht, 1983.
8. Galmiche, D., and Y. Salhi, *Sequent calculi and decidability for intuitionistic hybrid logic*, Information and Computation **209** (2011), 1447–1463.
9. Garg, D., V. Genovese and S. Negri, Countermodels from sequent calculi in multi-modal logics, In: Proc. LICS, 2012.
10. Marin, S., and L. Straßburger, Label-free modular systems for classical and intuitionistic logics, In: Proc. AiML, 2013.
11. Plotkin, G. D., and C. P. Stirling, A framework for intuitionistic modal logic, In: Halpern, J. Y. (ed.), Theoretical Aspects of Reasoning about Knowledge (1986).
12. Poggiolesi, F., The method of tree-hypersequents for modal propositional logic, In: Makinson, D., J. Malinowski and H. Wansing (eds.), Towards Mathematical Philosophy, Trends in Logic **28** (2009), 31–51.
13. Simpson, A., “The Proof Theory and Semantics of Intuitionistic Modal Logic”, Ph.D. thesis, Edinburgh University, Edinburgh, 1994.
14. Veloso, P. A. S., and S. R. M. Veloso, On graph refutation for relational inclusions, In: Rocca, S. R. D. and E. Pimentel (eds.), EPTCS **81** (2011), 47–62.
15. Veloso, P. A. S., S. R. M. Veloso and M. R. F Benevides, On graph calculi for multi-modal logics, In: Ayala-Rinc´on, M. and I. Mackie (eds.) LSFA 2014, 223–241, Bras´ılia, D. F., 2014.