Electronic Notes in Theoretical Computer Science 216 (2008) 3–29 

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

*a*-Logic With Arrows

Murdoch J. Gabbay1 Michael J. Gabbay2

**Abstract**

We present an extension of first-order predicate logic with a novel predicate ‘**at** *t*’ meaning intuitively “this term is a variable symbol”. We give simple sequent proof-rules for it, we demonstrate cut-elimination for the resulting logic, and we give a semantics for which the logic is sound and complete.

Because we can now make assertions about what would normally be considered an intensional property of a term (being a variable symbol) we can now express inside the logic, properties of its terms and predicates which would normally be external to the logic. We give axiomatisations in *a*-logic, including of the lambda- calculus, and discuss what relevance this might have to logic programming.

*Keywords:* First-order logic, lambda-calculus, soundness, completeness.

# Introduction

*a*-Logic extends classical First-Order Logic (FOL) with a predicate **at** *t* such that if *t* is *not* a variable symbol then **at** *t* is contradictory. We read **at** *t* as ‘*t* is a variable symbol’.

Let *a*, *b*, *c*, and so on, be variable symbols. A simple inductive definition of substitution on *abstract syntax* without binding is:

*a*[*a*:=*t*] *≡ t a/≡b ⇒ b*[*a*:=*t*] *≡ b f* (*a*1*,..., an*)[*a*:=*t*] *≡ f* (*a*1[*a*:=*t*]*,..., an*[*a*:=*t*]) Here *f* is a term-former and we write *s ≡ t* for ‘*s* and *t* are identical terms’, and *s /≡ t*

for ‘*s* and *t* are syntactically different terms’. For the first clause it is important that *a* is a variable symbol, and for the second clause it is important that *both a* and *b* are variable symbols. The third clause, as stated, ignores capture avoidance (if *f* is a binding term former) we assume that the reader is familiar with appropriate additional clauses/conventions to accommodate this.

1 [http://www.gabbay.org.uk](http://www.gabbay.org.uk/)

2 michael.gabbay(at)kcl.ac.uk Michael Gabbay gratefully acknowledges the support of the British Academy under the grant PDF/2006/509.

1571-0661Crown Copyright © 2008 Published by Elsevier B.V. Open access under [CC BY-NC-ND license](http://creativecommons.org/licenses/by-nc-nd/3.0/).

doi:10.1016/j.entcs.2008.06.031

We find that **at** gives just enough power to axiomatise the *λ*-calculus. We investigate the *λ*-calculus example in detail but by the end of the paper it should seem plausible that *a*-logic has broader applications.

*So what is new?* Plenty of other systems can express the syntax and reductions of the *λ*-terms. To our knowledge *a*-logic is unique in that it has this expressivity while still being a first-order (that is, *β*-reduction is not primitive) logic such that: variables of the object theory are variables of the logic *and λ* and application are just binary term-formers taking pairs of terms and giving terms.

We would like to say that again, but we need some (informal) terminology. Say a map is a ***shallow embedding*** from one system (logic or calculus, say) to another, when variables map to variables (with a ***deep embedding*** variables map to constants) [[2](#_bookmark42),[29](#_bookmark69)]; a map is ***compositional*** when the interpretation of an expression can be obtained directly by composing the interpretation of its parts in a syntax- directed manner. Say a system is ***first-order*** when its language of terms does not already contain *β*-reduction.

Our application of *a*-logic is unique in that it is to our knowledge the only compositional shallow embedding of the *λ*-calculus into a first-order logic.

The reader need not care about logic *or* about the *λ*-calculus. However, these are paradigmatic and they are the basis of logic and functional programming re- spectively. It has not previously been possible to directly (in the sense just given) map one to the other. We discuss potential applications and related work in the Conclusions.

*Outline of the paper. a*-logic is a straightforward classical first-order logic with a Boolean semantics. We define syntax in Subsection [2.2](#_bookmark0) and sequent rules in Subsec- tion [2.3](#_bookmark3). We axiomatise substitution in Subsection [3.1](#_bookmark12), then the untyped *λ*-calculus in Subsection [3.2](#_bookmark17), and finally we briefly consider other examples. Section [4](#_bookmark18) proves cut-elimination. Section [5](#_bookmark21) constructs a class of models and proves soundness and completeness. In the Conclusions we recap on related work, summarise, and men- tion possible future work. In Section [6](#_bookmark33) we build a concrete semantics for the axioms of the *λ*-calculus, incidentally proving consistency of the main axiomatisations in this paper.

The ‘*a*’ in ‘*a*-logic’ refers to the fact that the logic is for making statements about its variable symbols *a, b, c*. Any resemblance to an indefinite article is coincidental.

# *a*-logic

* 1. *The aim of a-logic*

Ultimately *a*-logic will be sensitive to its own syntax in addition to possessing a familiar denotational model theory. Truth in a model will therefore depend not only on the denotations of predicates, but also on their syntactic structure. For example the predicate ‘*x* is a variable’ will be true in a model not simply because of the denotation of ‘*x*’ but also because ‘*x*’ *is* a variable. Not every predicate is sensitive to term complexity in this way, some ignore it completely, others only partially. For example the predicate ‘*t*1 is a simplification of *t*2’, if true, ought to

remain true if *t*1 is replaced only by *simpler* terms with the same denotation, and if *t*2 is replaced only by *more complex* terms with the same denotation. To capture these intuitions formally, we assign to predicates an *arity* which indicates how it reacts to variation in the syntactic properties of its terms.

The syntactic property we are most interested in *a*-logic recognising is that of ‘being a variable’. The model theory of *a*-logic shall contain special elements, call them *atoms*, which represent variables in the denotation. But the variables of our syntax should still range over the whole domain of the model. So we use the predicate **at** *t*, which means that *t* is a variable symbol and is interpreted as and atom in the model. We can now extend our use of the term ‘atom’ to apply to variable symbols as well: a variable symbol *a* is an ***atom*** when we know that **at** *a*.

* 1. *Terms, directions, predicates*

Assume a countably infinite set *a, b, c ∈* A of variable symbols and some untyped language of abstract syntax trees. Write *s*, *t*, for arbitrary terms, built inductively from variable symbols and some countable set of ***term-formers*** of fixed arity ap- plied to terms. Write *t*[*a*:=*t'*] for the usual substitution on terms. We may write *a ∈ t* if *a* occurs in *t* and *a /∈ t* otherwise. [3](#_bookmark1)

We assume three ***directions*** *↑* ***up***, *↓* ***down***, and Ç ***up and down***. Where necessary, *d* will vary over directions. Assume some countable set of ***predicate constant symbols*** *p, q, r . . . ∈* P, each with an arity *δ* = (*d*1*,..., dn*) which is a sentence in *{↑, ↓,* Ç*}∗* (thus, a possibly empty list of directions). We write *p* : *δ* for ‘*p* ***has arity*** *δ*’. When *δ* has length *n* and some list of terms also has length *n*, we say that list has ***length appropriate to*** *δ*. We assume distinguished predicate constants:

* + 1. We call ~ : (*↑, ↓*) ***aequality*** (pronounced as ‘ayquality’).
    2. We call **at** : (*↓*) ***atom*** (‘is an atom’).
    3. We call *⊥* : () ***false*** or ***contradiction***.

We may write *ts* as shorthand for a list of terms *t*1*,..., tn*. Write *p*(*ts*) for a predicate constant applied to a list of terms of length appropriate to its arity.

Consider *p* : (*d*1*,..., dn*) and a list of terms (*t*1*,..., tn*). When for all *i* such that *a ∈ ti*, it is the case that *di ∈ {↑,* Ç*}*, we say *a* ***occurs up in*** *p*(*ts*) and we write *a↑p*(*ts*). Similarly when for all *i* such that *a ∈ ti* we have *di ∈ {↓,* Ç*}*, write *a↓p*(*ts*). When for all *i* such that *a ∈ ti* we have *di* = Ç, write *a*Ç*p*(*ts*).

The intuition of ~ is a formalisation of a reduction relation (like *β*-reduction, for example). The intuition of *a↓p*(*ts*) is a ‘subject reduction’ property, that if *s* ~ *t* then *p*(*ts*)[*a*:=*s*] implies *p*(*ts*)[*a*:=*t*]. If *a↑p*(*ts*) and *s* ~ *t* then *p*(*ts*)[*a*:=*t*] implies *p*(*ts*)[*a*:=*s*]. Finally, if *a*Ç*p*(*ts*) and *s* ~ *t* then *p*(*ts*)[*a*:=*s*] if and only if *p*(*ts*)[*a*:=*t*]. More on this in Subsection [2.4](#_bookmark5).

Intuitively, **at** *t* is true when *t* is a variable symbol. The arity (*↑, ↓*) of ~ is consistent with its intuition. **at** must have arity (*↓*) both to avoid inconsistencies

3 If the language of terms admits binding then *a ∈ t* when *a* occurs free in *t*.

(we see this later) and to remain true to the intuition. [4](#_bookmark4)

There is no interaction (yet) between directions and the terms, so *p*(*ts*) is always well-formed so long as *ts* has the right length. ***Predicates*** are generated by the grammar

*P* ::= *p*(*ts*) *|⊥| P ⊃ P | ∀a.P.*

*∀a.P* is binding, nothing else is. As is usual we write *∀a, b, . . . .P* as shorthand for *∀a.∀b....P* . We equate predicates up to *α*-equivalence; we will *not* talk about ‘occurring free’ or ‘occurring bound’. Consistent with our convention for terms we write *P ≡ Q* for ‘*P* and *Q* are identical predicates up to *α*-equivalence’.

Write *VP* for ***the variables occurring in*** *P* , defined by: *V p*(*t*1*,..., tn*) =

*V ti*, *V* (*P ⊃ Q*)= *VP ∪ V Q*, and *V* (*∀b.P* )= *VP\{b}*. We may write *a ∈ P* for

*a ∈ VP* , we read it ‘*a* ***occurs in*** *P* ’. Then *a /∈ P* means ‘*a* does not occur in *P* ’.

Write *P* [*a*:=*s*] for *P* with every instance of the variable *a* replaced by the term

*s* in the usual, capture-avoiding, manner.

* 1. *Contexts and judgements*

A ***(logical) context*** Γ is a finite set of predicates. We write *a ∈* Γ when *a ∈ P* for some *P ∈* Γ, and *a /∈* Γ otherwise.

A ***judgement*** is a pair of contexts which we write Γ *▶* Δ. When a context is on the right-hand side of a judgement we call it a ***cocontext***. The ***valid*** or ***derivable*** judgements are inductively defined by the rules in Figure [1](#_bookmark6).

In Figure [1](#_bookmark6) *a* and *b* are distinct variable symbols; bound by *∀*, free otherwise (note that the *a* in (**at L**) must be a variable symbol). Formulae in square brackets are *side-conditions* whose satisfaction can be decided just by examining syntax. (**at L**) is not mis-typed. It *eliminates* a variable.

Call Γ *▶* Δ a ***theorem*** when a derivation exists concluding in Γ *▶* Δ. If Γ = *∅*

write Γ *▶* Δ as *▶* Δ. If Δ = *∅* write Γ *▶* Δ as Γ *▶*.

A subset of the logic is classical predicate logic, so we use standard sugar such as writing *¬P* for *P ⊃ ⊥*, *P ∨ Q* for (*¬P* ) *⊃ Q*, *∃a.P* for *¬*(*∀a.¬P* ), *P ∧ Q* for

*¬*(*P ⊃* (*¬Q*)), and in general we shall use other well-known shorthands for classical equivalences.

* 1. *Comments on atoms*

(**at L**) states that atoms exist, and (**at R**) states that we cannot indirectly describe them (that is, by means of a complex term).

As can be seen from this derivation, assuming *t* is not a variable,

(**Ax**)

Γ*,* **at** *t ▶* **at** *t,* Δ

Γ*,* **at** *t ▶* Δ

(**at R**)

4 For example, if ~ models a literal reduction relation then we can interpret *t* ~ *a* as expressing that some complex term *t* reduces to an atom *a*. In that case we had better not allow the substitution of *t* for *a* in a (true) sentence such as **at** *a*, for that would yield the (potentially false) sentence **at** *t*.

Γ*,P ▶ P,* Δ

(**Ax**)

Γ*,P ▶ Q,* Δ

(*⊃* **R**)

Γ *▶ P ⊃ Q,* Δ

Γ *▶ P,* Δ Γ*,Q ▶* Δ

(*⊃* **L**)

Γ*,P ⊃ Q ▶* Δ

Γ *▶ P,* Δ [*a /∈* Γ*,* Δ] Γ *▶ ∀a.P,* Δ

(*∀***R**)

Γ*,P* [*a*:=*t*] *▶* Δ

Γ*, ∀a.P ▶* Δ

(*∀***L**)

Γ *▶ P,* Δ Γ*,P ▶* Δ

(**Cut**)

Γ *▶* Δ

Γ*, ⊥▶* Δ

(*⊥***L**)

Γ*,* **at** *a ▶* Δ [*a /∈* Γ*,* Δ] Γ *▶* Δ

(**at L**)

Γ *▶* **at** *t,* Δ [*t* not a variable] Γ *▶* Δ

(**at R**)

(~**R**)

Γ *▶ t* ~ *t,* Δ

Γ *▶ p*(*ts*)[*a*:=*t'*]*,* Δ [*a↓p*(*ts*)]

Γ*, t'* ~ *t ▶ p*(*ts*)[*a*:=*t*]*,* Δ

(~**L***↓*)

Γ *▶ p*(*ts*)[*a*:=*t*]*,* Δ [*a↑p*(*ts*)]

Γ*, t'* ~ *t ▶ p*(*ts*)[*a*:=*t'*]*,* Δ

(~**L***↑*)

Fig. 1. Derivation rules of *a*-logic with aequality

(**at R**) is like (*⊥***L**); ‘if *t* is a term then it is not a variable’. **at** *⟨a, a⟩▶* is a theorem and can be derived by (**Ax**) followed by (**at R**). (We assume a pair term-former

*⟨*-*,* -*⟩*.) There is no rule Γ *▶* **at** *a,* Δ. It would break Lemma [4.1](#_bookmark19) (the substitution lemma) which underlies the essential case of *∀* and which formalises the intuition ‘variable symbols represent unknown terms’.

We cannot conclude **at** *a* just because *a is* a variable symbol! **at** *a* in Γ represents a promise that *a* will never be instantiated to a (non-variable) term. This explains (**at R**) since there the promise has manifestly been broken, and also (**at L**) since there *a* is fresh from Γ and thus immune to any nasty things Γ may do to it (such as contain *a* = 0). We may call a variable symbol *a* of which we know **at** *a* an ***atom***.

Define *a↑P* , *a↓P* , and *a*Ç*P* by:

*a*Ç*P a↑P*

*a*Ç*P a↓P*

*a↑P a↓Q a↓*(*P⊃Q*)

*a↓P a↑Q a↑*(*P⊃Q*)

*a*Ç*P a*Ç*Q a*Ç(*P⊃Q*)

*a↑P a↑∀a.P*

*a↓P a↓∀a.P*

*a*Ç*P a*Ç*∀a.P*

**Theorem 2.1** ~ *is transitive and reflexive; s* ~ *t, t* ~ *u ▶ s* ~ *u and ▶ s* ~ *s*

*are derivable.*

**Proof.** The latter is simply an instance of (~**R**), the former follows by this deriva- tion (noting that *a↓*(*s* ~ *a*)):

*s* ~ *t ▶ s* ~ *a*[*a*:=*t*]

(**Ax**)

(~**L***↓*)

*s* ~ *t, t* ~ *u ▶ s* ~ *a*[*a*:=*u*]

~ behaves like the transitive reflexive closure of a reduction/rewrite relation. By Theorem [2.2](#_bookmark8) below if *a↓P* then *P* satisfies ‘subject reduction’; if *P* [*a*:=*s*] and *s* ~ *t* then *P* [*a*:=*t*]. But then in *P ⊃ Q* (where *Q* does not mention *a*) the direction of the ‘subject reduction’ is reversed, thus we need *a↑*(*P ⊃ Q*). Similar issues arise with ~ itself. For example if (*a* ~ *t*)[*a*:=*s*] and *s'* ~ *s* then (*a* ~ *t*)[*a*:=*s'*]. This is why ~ has arity (*↑, ↓*).

**Theorem 2.2** *The following two rules are derivable:*

Γ *▶ P* [*a*:=*t'*]*,* Δ (*a↓P* )

Γ*, t'* ~ *t ▶ P* [*a*:=*t*]*,* Δ

(~**L***↓*)

Γ *▶* Δ*,P* [*a*:=*t*] (*a↑P* )

Γ*, t'* ~ *t ▶* Δ*,P* [*a*:=*t'*]

(~**L***↑*)

**Proof.** It is not hard to check, by induction on *P* , that directions propagate point- wise but get flipped if they are in negative position (the left-hand side of an impli- cation) and that *a*Ç*P* precisely when *a↑P* and *a↓P* .

We now work by induction on *P* . The atomic case follows immediately from (~**L***↑*)*,* (~**L***↓*) and the definition of *a↓P* in the case where *P* is atomic (see page [3](#_bookmark2)). The induction cases are routine. For example suppose *P* is (*Q ⊃ R*). We must show that

* + 1. if *a↓*(*Q ⊃ R*) and Γ *▶* (*Q ⊃ R*)[*a*:=*t'*]Δ then Γ*, t'* ~ *t ▶* (*Q ⊃ R*)[*a*:=*t*]*,* Δ
    2. if *a↑*(*Q ⊃ R*) and Γ *▶* (*Q ⊃ R*)[*a*:=*t*]*,* Δ then Γ*, t'* ~ *t ▶* (*Q ⊃ R*)[*a*:=*t'*]*,* Δ

The antecedent of case (i) can be only if we have *a↑Q* and *a↓R*, and the antecedent for case (ii) can occur only if *a↓Q* and *a↑R*.

For case (i) we sketch a derivation as follows:

Γ*, Q*[*a*:=*t*] *▶ Q*[*a*:=*t*]*,* Δ

(**Ax**)

*Ind.Hyp.,* (~**L***↑*)

Γ*, R*[*a*:=*t'*] *▶ R*[*a*:=*t'*]*,* Δ

(**Ax**)

*Ind.Hyp.*(~**L***↓*)

Γ*, t'* ~ *t, Q*[*a*:=*t*] *▶ Q*[*a*:=*t'*]*,* Δ

Γ*, t'* ~ *t, R*[*a*:=*t'*] *▶ R*[*a*:=*t*]*,* Δ

(*⊃* **L**)

Γ*, t'* ~ *t,* (*Q ⊃ R*)[*a*:=*t'*]*, Q*[*a*:=*t*] *▶ R*[*a*:=*t*]*,* Δ

Γ*, t'* ~ *t,* (*Q ⊃ R*)[*a*:=*t'*] *▶* (*Q ⊃ R*)[*a*:=*t*]*,* Δ

(*⊃* **R**)

We can now derive Γ*, t'* ~ *t ▶ P* [*a*:=*t*]*,* Δ from the assumption that Γ *▶* (*Q ⊃*

*R*)[*a*:=*t'*]*,* Δ and (**Cut**).

The relevant derivation for case (ii) is easily obtained from the derivation above by swapping *t* with *t'* and relabelling the applications of (~**L***↑*) and (~**L***↓*) accord- ingly.

The standard derivation rules for equality are: [5](#_bookmark9)

5 A familiar alternative to (=**L**) is this:

Γ*,P* [*a*:=*t'*] *▶* Δ

Γ*, t'* = *t, P* [*a*:=*t*] *▶* Δ

(=**L***'*)

Γ *▶ t* = *t,* Δ

(=**R**)

Γ *▶ P* [*a*:=*t'*]*,* Δ

Γ*, t'* = *t ▶ P* [*a*:=*t*]*,* Δ

(=**L**)

If we make sure that *a*Ç*P* , then ~ behaves just like an equality — at least for that

*a*. Also, *a*-logic predicates can express a notion of equality:

**Definition 2.3** Write *t* = *u* in *a*-logic as shorthand for *t* ~ *u ∧ u* ~ *t*.

**Theorem 2.4** (=**R**) *and* (=**L**) *are derivable in a-logic.*

**Proof.** If neither *a↑p*(*ts*) nor *a↓p*(*ts*) we view *p*(*ts*) in terms of two variables fresh atoms *a'* and *a''*, one occurring exclusively up in *p*(*ts*), the other occurring ex- clusively down. That is, we view *p*(*ts*) as *p'*(*ts*)[*a'*:=*a, a''*:=*a*] where *a'↓p'*(*ts*) and *a''↑p'*(*ts*), and where *a /∈ p'*(*ts*). It is then easy to verify that with one use each of (~ **L***↑*) and (~ **L***↓*) we can obtain the effect of (=**L**). (=**R**) is easy.

So *a*-logic combines elements of computation, given by ~, of logic, given by the evident logical apparatus, and of something quite unusual, given by **at** .

# Expressivity

We are now ready to write down some axioms:

* 1. *Substitution*

**Definition 3.1** Suppose a ternary term-former *⟨⟩*. We usually write *⟨⟩*(*s, u, t*) as *s⟨u'→t⟩*. Assume a binary predicate # : (*↓, ↓*) ***freshness***. Then an *a*-logic theory of substitution is given by the axioms in Figure [2](#_bookmark14).

We intend # to express a notion of ‘not free in’. This notation and intuition is inherited from nominal techniques [[18](#_bookmark58)], though the properties of # are here expressed as axioms in *a*-logic, whereas they are built in as primitive to the nominal framework. We intend *⟨⟩* to express a notion of ‘capture-avoiding substitution for atoms’; a related nominal treatment is also available [[17](#_bookmark57)].

**Remark 3.2** Note, in Figure [3](#_bookmark15) that *a*#*t* implies **at** *a*. We use this for example when we omit an assumption **at** *a* in a number of axioms, e.g. (*'→***comm**), but we still give *a* the name ‘*a*’.

A remark on syntax is particularly important:

**Remark 3.3** *v⟨s'→t⟩* is valid syntax whether or not *s is* a variable symbol. *⟨⟩* is just a ternary term-former. *⟨⟩* is distinct from substitution on terms *v*[*a*:=*t*]. For example *c*[*a*:=*b*] *≡ c* but *c⟨a'→b⟩ /≡ c* and furthermore *▶ c⟨a'→b⟩* ~ *c* is not derivable. *v*[*s*:=*t*] is not well-defined unless *s ≡ a* for some variable symbol *a*. *v⟨s'→t⟩* is a valid term always, but only if we have assumed **at** *s* can we prove anything useful about *v⟨s'→t⟩* from the axioms in Figure [2](#_bookmark14). However, assuming **at** *s* of *s* that is not

(*'→*#) *∀a, v.* *a*#*v ⊃ ∀b.*(**at** *b ⊃ v⟨a'→b⟩* ~ *v*)

(*'→***aa**) *∀a, v.*(**at** *a ⊃ v⟨a'→a⟩* ~ *v*)

(*'→***at**) *∀a, t.*(**at** *a ⊃ a⟨a'→t⟩* ~ *t*)

(*'→***ren**) *∀a, b, u, v.* (**at** *a ∧ b*#*v*) *⊃ v⟨a'→b⟩⟨b'→u⟩* ~ *v⟨a'→u⟩*

(*'→***comm**) *∀a, b, v, t, u.* (**at** *b ∧ a*#*b ∧ a*#*u*) *⊃ v⟨a'→t⟩⟨b'→u⟩* ~ *v⟨b'→u⟩⟨a'→t⟨b'→u⟩⟩*

Fig. 2. *a*-logic axioms for substitution

(#**aa**) *∀a.*(*¬a*#*a*)

(#**at** ) *∀a, t.*(*a*#*t ⊃* **at** *a*)

(# ~) *∀a, b.* (**at** *a ∧* **at** *b*) *⊃* (*a* ~ *b ∨ a*#*b*)

Fig. 3. *a*-logic axioms for freshness

(*λ***ref** ) *∀a, b, v.*(**at** *a ∧ b*#*v ⊃ λa.v* = *λb.v⟨a'→b⟩*) (*λβ*) *∀a, v, t.*(**at** *a ⊃* (*λa.v*)*t* ~ *v⟨a'→t⟩*)

Fig. 4. *a*-logic axioms for *α*-equality and for *β*-reduction

a variable symbol leads to immediate contradiction by (**at R**). Therefore in practise we only have occasion to usefully write down terms such as *t⟨a'→u⟩*.

We can now give an answer to the question “why ~ not =?”. Assume the axioms of Figure [2](#_bookmark14), except replacing ~ with = from Definition [2.3](#_bookmark10), which we proved in Theorem [2.4](#_bookmark11) has the properties we expect of equality. For example axiom (*'→***at**) becomes *∀a, t.*(**at** *a ⊃ a⟨a'→t⟩* = *t*). Call this set of axioms Σ*'*. Then:

**Theorem 3.4** Σ*' ▶ is derivable. In words: “*Σ*' is inconsistent in a-logic”.*

**Proof.** Note that all the axioms of Σ*'* are closed and that Σ*',* **at** *a ▶ a⟨a'→a⟩* = *a* is derivable using the replaced version of (*'→***aa**). Using (**Cut**) and (=**L**) also Σ*',* **at** *a ▶* **at** (*a⟨a'→a⟩*) is derivable and using (**at R**) we obtain Σ*',* **at** *a ▶*. Finally by (**at L**) we derive Σ*' ▶*.

So **at** in *a*-logic interacts with equality, and seems to require it to be directional. In Section [6](#_bookmark33) we build a nontrivial model for all the axioms of Figures [2](#_bookmark14), [3](#_bookmark15) and [4](#_bookmark16).

By a soundness result which we prove later in Theorem [6.7](#_bookmark39), this demonstrates that any subset or weakening of these axioms is neither inconsistent nor trivial.

**Remark 3.5** The axioms of Figure [3](#_bookmark15) interact to yield some further theorems of substitution and freshness. For example, using (~ **R**) it follows that

*b*#*a, a* ~ *b ▶ b* ~ *b*

and then using (#**aa**) we can derive that *b*#*a ▶ ¬*(*a* ~ *b*). Then using (# ~) we can derive

**at** *a,* **at** *b, b*#*a ▶ a*#*b*

so freshness is a symmetric relation when between atoms.

**Remark 3.6** A corollary of Theorem [6.7](#_bookmark39) is that it is consistent to strengthen (*'→***comm**) to

*∀a, b, v, t, u.* (**at** *b ∧ a*#*b ∧ a*#*u*) *⊃ v⟨a'→t⟩⟨b'→u⟩* = *v⟨b'→u⟩⟨a'→t⟨b'→u⟩⟩*

This may be appropriate in some examples, for example such in the *λ*-calculus of Subsection [3.2](#_bookmark17).

Axiomatisations of substitution exist though we are aware of no authoritative source. Two examples are Crabb´e [[10](#_bookmark50), p.2] and Salibra [[23](#_bookmark63), p.6]. Crabb´e assumes a function symbol which we can identify with #; Salibra gives a more algebraic ax- iomatisation but the domain *must* be a model of the *λ*-calculus so he expresses *a*#*s* (*a* is not free in *s*) by writing *s* as (*λx.s*)*a*. Salibra’s and Crabb´e’s axiomatisations are almost equivalent; the authors do not cite one another but the interested reader can note that (*λx.s*)*t* in Salibra corresponds to *s*[*a*:=*t*] in Crabb´e [[10](#_bookmark50), Proposition 3.1]. We must remove Salibra’s axiom *β*6, which is *β*-reduction. The first author with Mathijssen has recently investigated an axiomatisation of substitution using nominal algebra [[16](#_bookmark56),[17](#_bookmark57)]. Nominal algebra makes more structure, such as freshness, primitive to the logic; making a formal connection with *a*-logic is future work.

* 1. *λ-calculus*

**Definition 3.7** Assume the term-former substitution *⟨⟩* and predicate constant freshness # : (*↓, ↓*) from Subsection [3.1](#_bookmark12). Also assume binary term-formers ***applica- tion*** *·* and ***lambda*** *λ*. Write *·*(*t', t*) as *t' · t*. Write *λ*(*s, t*) as *λs.t*. An *a*-logic theory of the *λ*-calculus is given by the axioms in Figures [2](#_bookmark14), [3](#_bookmark15) and [4](#_bookmark16).

Note (as observed of *⟨⟩* in Remark [3.3](#_bookmark13)) that application and *λ* are merely term formers; *λs.t* is a valid term for any *s*, but the axioms permit nothing of interest to be derived about it, unless **at** *s* is also derivable.

We can add an axiom for extensionality to those of Figure [4](#_bookmark16):

*∀b.∀s.***at** *b ⊃* (*λb.s*)*b* ~ *s.*

There are other interesting properties we might consider investigating in future work. For example:

* A unary predicate symbolwith axiom *s ⇔ ∀a.***at** *a ⊃ a*#*s*. The intuition is ‘*s* is closed’.
* A predicate ‘the free atoms of *s* are precisely *b*’ axiomatised by (*¬b*#*s*) *∧ ∀a.***at** *a∧*

*a /*= *b ⊃ a*#*s*.

* A predicate ‘the free atoms of *s* are at most *b*’ axiomatised by *∀a.***at** *a ∧ a /*= *b ⊃*

*a*#*s*.

These are in the spirit of suggestions already made for nominal rewriting in [[11](#_bookmark51), Subsection 9.2] though the technical details are very different.

The first author was instrumental in developing so-called Nominal techniques, including Nominal Logic [[22](#_bookmark59),[14](#_bookmark54)]. These have similar applications to *a*-logic and they have a freshness predicate #. However they are inconsistent with the axiom of choice; for example Nominal Logic is inconsistent with a unary term-former *f* such that (*f s*)#*s* always. It seems that we *can* axiomatise such an *f* in *a*-logic, as

*∀s.∃b.fs* ~ *b ∧ b*#*s*. The use of ~ helps to avoid contradictions and this might be useful in some applications of nominal techniques.

# Cut-elimination

In this section we prove that *a*-logic (without additional axioms and term formers for substitution) satisfies Cut-Elimination. If Γ = *{G*1*,..., Gn}* write Γ[*a*:=*t*] for

*{G*1[*a*:=*t*]*,..., Gn*[*a*:=*t*]*}*. Similarly for Δ[*a*:=*t*].

Call the total number of instances of derivation rules in a derivation, its ***depth***.

**Lemma 4.1 (Substitution Lemma)** *If* Γ *▶* Δ *has a derivation* Π*, then* Γ[*a*:=*t*] *▶*

Δ[*a*:=*t*] *has a derivation* Π[*a*:=*t*] *that is no deeper than* Π*.*

**Proof.** By induction on the depth of derivations.

We consider some cases:

* The case of (**at R**). If *v* is not a variable in Γ *▶* **at** *v,* Δ then *v*[*a*:=*t*] is not a variable in Γ[*a*:=*t*] *▶* **at** *v*[*a*:=*t*]*,* Δ[*a*:=*t*] so substitution instances of instances of (**at R**) are still instances of (**at R**).
* The case of (**at L**). Suppose the sequent Γ *▶* Δ is derived from the sequent Γ*,* **at** *b ▶* Δ by (**at L**). Then *b* is a variable symbol such that *b /∈* Γ*,* Δ. We need to show that Γ[*a*:=*t*] *▶* Δ[*a*:=*t*] is derivable.

By the inductive hypothesis, we have that Γ[*b*:=*b'*]*,* **at** *b*[*b*:=*b'*] *▶* Δ[*b*:=*b'*] is derivable, for any other variable *b'*. Choosing *b'* so that *b' /∈* Γ*,* Δ*,t* it follows that the sequent Γ*,* **at** *b' ▶* Δ is derivable, and the derivation is no deeper.

By the inductive hypothesis again

Γ[*a*:=*t*]*,* (**at** *b'*)[*a*:=*t*] *▶* Δ[*a*:=*t*]

is derivable. Now, *b'* was chosen so that (**at** *b'*)[*a*:=*t*] *≡* **at** *b'*. So we can use (**at L**) to derive Γ[*a*:=*t*] *▶* Δ[*a*:=*t*] as required.

* The rules for ~ are non-standard but it is not hard to verify that substitution

in a valid instance of a rule is still a valid instance, just as we would do for the usual equality rules.

For example in the case of (~**L***↓*) if we have

Π

*·*

*·*

*·*

Γ *▶ p*(*ts*)[*b*:=*t'*]*,* Δ [*b↓p*(*ts*)]

Γ*, t'* ~ *t ▶ p*(*ts*)[*b*:=*t*]*,* Δ

. . . then by induction hypothesis we have:

Π[*a*:=*s*]

*·*

*·*

*·*

(~**L***↓*)

Γ[*a*:=*s*] *▶ p*(*ts*)[*b*:=*t'*[*a*:=*s*]]*,* Δ[*a*:=*s*] [*b↓p*(*ts*)]

Γ[*a*:=*s*]*,* (*t'* ~ *t*)[*a*:=*s*] *▶ p*(*ts*)[*b*:=*t*[*a*:=*s*]]*,* Δ[*a*:=*s*] Other cases are routine.

**Lemma 4.2 (Weakening)** *If* Γ *▶* Δ *then* Γ*,* Γ*' ▶* Δ*,* Δ*'.*

(~**L***↓*)*.*

**Proof.** By induction on the derivation of Γ *▶* Δ. We may need to rename variables generated by (**at L**), or (*∀***R**) to avoid clashes with variables in Γ*'* or Δ*'*. For example if the derivation ends with (**Ax**) then the weakened conclusion is itself derivable by (**Ax**), if the derivation ends with (*∀***R**):

Π

*·*

*·*

*·*

Γ *▶* Δ

. . . then by the induc- tion hypothesis we

Π[*a*:=*c*]

*·*

*·*

*P,*

(*∀***R**)(*∗*)

have. . .

*·*

Γ*,* Γ*' ▶ P* [

] Δ Δ*'*

Γ *▶ ∀a.P,* Δ

*a*:=*c , ,*

where Π[*a*:=*c*] is obtained using Lemma [4.1](#_bookmark19) by replacing *a* in Π by some atom *c* that does not occur in Γ*,* Γ*',* Δ or Δ*'*. We know that Π[*a*:=*c*] really is a derivation because *a* is not free in Γ or Δ. From this we can derive Γ*,* Γ*' ▶ ∀a.P,* Δ*,* Δ*'* by an application of (*∀***R**) (recall from Subsection [2.2](#_bookmark0) that we equate predicates up to *α*-equivalence). The remaining cases follow by familiar application of the induction hypothesis.

Γ *▶ P,* Δ Γ*,P ▶* Δ

For any instance of (**Cut**): (**Cut**)

Γ *▶* Δ

say its ***degree*** is the total

number of instances of the symbols *⊥, ⊃, ∀* in *P* ; and say its ***rank*** is the depth of the derivation of its conclusion.

**Theorem 4.3** (**Cut**) *is an admissible rule in the system without it.*

**Proof.** By induction, lexicographically, on (*d, r*) where *d* is the degree and *r* is the rank of the earliest cut, counting from the leaves of the derivation down to

the conclusion. The proof is standard for first-order logic with equality and uses weakening, and the substitution lemma for the essential cases. (**at L**) and (**at R**) have no essential case. Cuts are commuted in the usual way with the rules, including (**at L**) and (**at R**), to lower their rank forming essential cases where they are reduced to cuts of lower degree.

Π2

*·*

*·*

Π1 *·*

*·* **at** *a,* Γ*,P ▶* Δ

*·*

*·*

(**at L**)

Π*'*

*·*1

*·*

*·*

=*⇒* **at** *a,* Γ *▶*

*P,* Δ

Π2

*·*

*·*

*·*

**at** *a,* Γ*,P ▶* Δ

(**Cut**)

Γ *▶ P,* Δ

Γ *▶* Δ

Γ*,P ▶* Δ

(**Cut**)

**at** *a,* Γ *▶* Δ

(**at L**)

Γ *▶* Δ

Here we use Weakening (proved above) to extend Π1 to Π*'* by adding **at** *a* to the

1

premise of every sequent in Π1. Since *a /∈* Γ*,* Δ*,P* we can rename all terms in Π1 so that *a* does not occur at all in it, and then we can add **at** *a* without invalidating any rule applications in Π. The essential (and commutation) cases for the remaining

connectives are familiar, and with one exception the essential cases for ~ are also easy.

Γ *▶ t* ~ *t,* Δ

(~**R**)

Γ *▶* Δ

Π

*·*

*·*

*·*

Γ *▶* Δ

Γ*,t* ~ *t ▶* Δ

(~**L***↓*)

Π

*·*

=*⇒*

*·*

*·*

Γ *▶* Δ

It is worth stating explicitly the problematic commutation case for aequality. This is the case where an aequality rule applies to the cut formula itself. Suppose *a↓p*(*ts*) and we have

Π*'*

*·*

*·*

*·*

Γ *▶ p*(*ts*)[*a*:=*t'*]*,* Δ

Γ*, t'* ~ *t ▶ p*(*ts*)[*a*:=*t*]*,* Δ

(~**L***↓*)

Π

*·*

*·*

*·*

Γ*, p*(*ts*)[*a*:=*t*] *▶* Δ

(**Cut**)

Γ*, t'* ~ *t ▶* Δ

then (by the other commutation cases) we can assume that the right premise of such a (**Cut**) where the cut formula is atomic, is actually the conclusion of (**Ax**), (*⊥***L**), (~ **R**) or (**at R**) (that is, we can assume Π is empty).

If the right premise of the (**Cut**) is derived by (*⊥***L**) or (~ **R**), or if the predicate relevant to its derivation is in Γ or Δ (and is not *p*(*ts*)[*a*:=*t*]), then the conclusion of the (**Cut**) can be derived directly by the rule that derived Γ*, p*(*ts*)[*a*:=*t*] *▶* Δ.

If the right premise of the (**Cut**) is derived from (**Ax**) so that Δ is Δ*', p*(*ts*)[*a*:=*t*]

then we may permute the derivation to Π*'*

*·*

*·*

*·*

(**Ax**)

Γ *▶ p*(*ts*)[*a*:=*t'*]*,* Δ Γ*, p*(*ts*)[*a*:=*t'*] *▶ p*(*ts*)[*a*:=*t'*]*,* Δ*'*

(**Cut**)

Γ *▶ p*(*ts*)[*a*:=*t'*]*,* Δ*'*

Γ*, t'* ~ *t ▶* Δ

(~**L***↓*)

The twins to these, where *a↑p*(*ts*), is similar.

**Lemma 4.4** *If* Γ *▶* Δ*, where* Γ *and* Δ *do not contain* **at** *or* ~ *and where the ﬁnal rule application is* (**at L**) *or* (**at R**) *to the formula* **at** *a, then a (shorter) derivation can be found without this application.*

**Proof.** By induction on the length of a cut-free derivation that Γ *▶* Δ. Suppose (**at L**) or (**at R**) follows to a derivation of length 1. Since Γ and Δ do not contain

~ the derivation must be a single application of (**Ax**) or (*⊥***L**). In the first case the derivation must be of one of these forms:

Γ*',* **at** *a, P ▶ P,* Δ*'* Γ *▶* Δ

(**Ax**)

(**at L**)

or Γ*',P ▶ P,* **at** *t,* Δ*'* Γ *▶* Δ

(**Ax**)

(**at R**)

where Γ = Γ*' ∪{P}* and Δ = Δ*' ∪{P}*. This can be replaced by a single application of (**Ax**). The case for (*⊥***L**) is similar.

Now suppose the derivation is of length *>* 1. Then the rule preceding the final application of (**at L**) cannot be any ~ rule (for then Γ would contain ~). Therefore the preceeding rule must be a *FOL* rule. Then we permute the application of (**at L**) or (**at R**) with the FOL rule (so the **at** rule applies to the premises of the FOL rule). We can then apply the induction hypothesis. For example:

Γ*,* **at**

Π

*·*

*·*

*·*

*a, P*

*▶ Q,* Δ

. . . gets permuted

Γ*,* **at**

Π

*·*

*·*

*·*

*a, P*

*▶ Q,* Δ

(*⊃***R**)

Γ*,* **at** *a ▶ P ⊃ Q,* Δ

(**at L**)

Γ *▶ P ⊃ Q,* Δ

to. . .

Γ*,P ▶ Q,* Δ Γ *▶ P ⊃ Q,* Δ

(**at L**)

(*⊃***R**)

and the induction hypothesis applies to the derivation up to the application of (**at L**).

**Corollary 4.5** *a-logic is a conservative extension of FOL without identity.*

**Proof.** Suppose Γ *▶* Δ, in the language of FOL without identity, is derivable in *a*-logic. We shall argue it is also derivable in FOL without identity. We know that Γ *▶* Δ can be derived in a cut-free derivation. If this derivation makes no use of the rules for **at** and ~ then there is nothing to prove, otherwise our argument proceeds by induction on the length of the derivation. If the final rule is a FOL rule then

the result follows by induction hypothesis on its premises. Given the conditions on Γ and Δ, the only other possibility for the final rule is that it is (**at L**) or (**at R**) , in this case the result follows by Lemma [4.4](#_bookmark20).

# Models of *a*-logic

* 1. *Semantic deﬁnitions and soundness*

An *a****-domain*** D is a tuple (*|*D*|, I*D*, ≤*D) — we may write (*|*D*|,I, ≤*) if D is understood

— such that:

* *|*D*|* is a set.

We call it the ***underlying set*** of D. We may write *x ∈* D for *x ∈ |*D*|*.

* (*|*D*|, ≤*) is a poset on *|*D*|*.

That is, *≤* is a transitive reflexive asymmetric binary relation on *|*D*|*.

* *I ⊆ |*D*|* is ***down-closed*** with respect to *≤*. That is, for any *x, x' ∈ |*D*|*, if *x' ∈ I*

and *x ≤ x'* then *x ∈ I*

A ***valuation*** *ς* to D is a function from the set of variable symbols A to *|*D*|* such that *ς*(*a*) *∈ I* for at least one *a ∈* A. Write *ς{a'→x}* for the valuation such that *ς{a'→x}*(*b*)= *ς*(*b*) for all *b* other than *a*, and *ς{a'→x}*(*a*)= *x*.

We need some notation. Write *|*D*|n* for the set of *n*-tuples (*x*1*,..., xn*) of elements *xi ∈ |*D*|* for 1 *≤ i ≤ n*. *|*D*|*0 contains just the 0-tuple (). Call a function *f* from *|*D*|n* to *|*D*|* ***monotone*** when

if *xi ≤ x'* then *f* (*x*1*,..., xi,..., xn*) *≤ f* (*x*1*,..., x',..., xn*).

*i* *i*

A ***model* [**- **]** extends an *a*-domain D with the following information:

* For each term-former *f* of arity *n* a monotone function **[***f* **]** from *|*D*|n* to *|*D*|* such that **[***f* ]](*x*1*,..., xn*) */∈ I* for all *x*1*,..., xn*.

Intuitively, the result of applying the interpretation of a term-former to some arguments must never be the interpretation of a variable symbol. This reflects the intuition that **at** *t* is false if *t* is *not* a variable symbol.

* For each predicate constant symbol *p* of arity (*d*1*,..., dn*) other than **at** and ~, a set **[***p*]] *⊆ |*D*|n* such that if *xi ≤ x'* then:

*i*

* + If *di* = *↑* and (*x*1*,..., xi,..., xn*) *∈* [[*p* **]** then (*x*1*,..., x',..., xn*) *∈* [[*p*]].

*i*

* + If *di* = *↓* and (*x*1*,..., x',..., xn*) *∈* [[*p* **]** then (*x*1*,..., xi,..., xn*) *∈* [[*p*]].

*i*

* + If *di* = Ç then (*x*1*,..., xi,..., xn*) *∈* [[*p* **]** if and only if (*x*1*,..., x',..., xn*) *∈* [[*p*]]. Comparing these conditions with (~**L***↓*) and (~**L***↑*) from Figure [1](#_bookmark6) we see that

*i*

*≤* models ~, and we make this intuition precise below.

We extend the model **[**- **]** to an ***interpretation*** of terms inductively by [[*a*]]*ς* = *ς*(*a*) **[***f* (*t*1*,..., tn*)]]*ς* = [[*f* ]]([[*t*1]]*ς ,...,* [[*tn*]]*ς* )

as is standard.

Given a valuation *ς* and a **[**- **]** we define a notion of ***validity*** in the model (for the valuation) as follows:

* [[*⊥*]]*ς* is invalid.
* [[*P ⊃ Q*]]*ς* is valid when **[***P* ]]*ς* is invalid or **[***Q*]]*ς* is valid.
* [[*∀a.P* ]]*ς* is valid when **[***P* ]]*ς{a'→x}* is valid for all *x ∈* D.
* For *p /∈ {*~*,* **at** *}*, [[*p*(*t*1*,..., tn*)]]*ς* is valid when ( **[***t*1]]*,...,* [[*tn*]])*ς ∈* [[*p*]].
* [[*t*1 ~ *t*2]]*ς* is valid when **[***t*2]]*ς ≤* [[*t*1]]*ς* .
* [[**at** *t*]]*ς* is valid when **[***t*]]*ς ∈ I*.

We write that **[**Γ **]***ς* is valid when **[***G*]]*ς* is valid for all *G ∈* Γ, and we extend validity to judgements Γ *▶* Δ by: **[**Γ *▶* Δ]]*ς* is valid when either

* [[*G*]]*ς* is not valid for some *G ∈* Γ, or
* [[*D*]]*ς* is valid for some *D ∈* Δ.

Finally we write

[[Γ *▶* Δ **]** is valid when **[**Γ *▶* Δ]]*ς* for all valuations *ς*. Lemmas [5.1](#_bookmark23) and [5.2](#_bookmark24) are technical results which are useful later:

**Lemma 5.1** [[**at** *t*]]*ς is valid if and only if t is a variable symbol and ς*(*t*) *∈ I.*

**Proof.** By definition, **[at** *t*]]*ς* is valid when **[***t*]]*ς ∈ I* and by the conditions on **[**- **]** and *ς*, [[*t*]]*ς* cannot be in *I* if *t* is a constant or a complex term (as no function symbol is interpreted to have values in *I*).

**Lemma 5.2** (i) **[***v*[*a*:=*t*]]]*ς* = [[*v*]]*ς{a'→*[[*t*]] *}.*

*ς*

(ii) **[***P* [*a*:=*t*]]]*ς is valid if and only if* [[*P* ]]*ς{a'→*[[*t*]] *} is valid.*

*ς*

**Proof.** The first part is by an easy induction on syntax. The second part is also by an easy induction, we consider only one case:

[[**at** (*v*[*a*:=*t*]) **]***ς* is valid if and only if **[***v*[*a*:=*t*]]]*ς ∈ I*, and by part (i) this is the

case if and only if **[***v*]]*ς{a'→*[[*t*]]

*∈ I*.

*ς*

*}*

Other cases are similar.

**Lemma 5.3 (**(*∀***L**)**)** (i) *If* [[*∀a.P* ]]*ς is valid then* [[*P* [*a*:=*t*]]]*ς .*

(ii) *If* [[Γ*,P* [*a*:=*t*] *▶* Δ]]*ς is valid then* [[Γ *∧ ∀a.P ▶* Δ]]*ς .*

**Proof.** The first part is direct from Lemma [5.2](#_bookmark24) and the definition of **[***∀a.P* ]]*ς* . The second part is routine from the definition, using the first part.

**Lemma 5.4** *Suppose that a /∈ t, a /∈ G and ς*(*b*)= *ς'*(*b*) *for all other b ∈* A*. Then*

1. **[***t*]]*ς* = [[*t*]]*ς'*
2. **[***G*]]*ς is valid iff* [[*G*]]*ς' is valid.*

*As an easy corollary, if a /∈* Γ*,* Δ *and ς*(*b*)= *ς'*(*b*) *for all other b ∈* A *then* [[Γ *▶* Δ]]*ς*

*if and only if* [[Γ *▶* Δ]]*ς' .*

**Proof.** We work by induction on *t* for (i) and then on *G* for (ii). We consider only two cases:

* The case *G ≡* **at** *t*. Suppose **[at** *t*]]*ς* is valid. By Lemma [5.1](#_bookmark23) *t* is a variable symbol and *ς*(*t*) *∈ I*. *t ≡ a* is impossible because we assumed *a /∈ t*. Therefore *ς'*(*t*) is valid. The result follows.

The reverse implication is similar.

* The case *G ≡ ∀b.P* . We can assume *b* is distinct from *a* because we equate syntax up to *α*-conversion. **[***∀b.P* ]]*ς* is valid when **[***P* ]]*ς{b'→x}* for all *x ∈* D. The result follows by inductive hypothesis.

**Lemma 5.5 (**(*∀***R**)**)** *If* [[Γ *▶ P,* Δ]] *and a /∈* Γ*,* Δ *then* [[Γ *▶ ∀a.P,* Δ]]*.*

**Proof.** Fix *ς*. We now reason by cases:

* Fix some *G ∈* Γ. By Lemma [5.4](#_bookmark26) [[*G*]]*ς* is not valid if and only if **[***G*]]*ς'* is not valid, for any other *ς'* such that *ς*(*b*) = *ς'*(*b*) for all *b* other than *a*. Therefore [[Γ *▶ ∀a.P,* Δ]]*ς* is valid.
* Fix some *D ∈* Δ. If **[***D*]]*ς* is valid then by Lemma [5.4](#_bookmark26) also **[**Γ *▶ ∀a.P,* Δ]]*ς* is valid.
* Suppose **[***G*]]*ς* is valid for all *G ∈* Γ and suppose **[***D*]]*ς* is not valid for all *D ∈* Δ. Then **[***P* ]]*ς* is valid. By Lemma [5.4](#_bookmark26) [[*P* ]]*ς{a'→x}* must be valid for all *x ∈* D. Therefore **[***∀a.P* ]]*ς* is valid and so **[**Γ *▶ ∀a.P,* Δ]]*ς* is valid as required.

**Lemma 5.6 (**(**at R**)**,**(**at L**)**)** (i) *If* [[Γ *▶* **at** *u,* Δ]]*ς is valid and u is not a variable symbol then* [[Γ *▶* Δ]]*ς is valid.*

(ii) *If* [[Γ*,* **at** *a ▶* Δ]]*ς is valid and* [[*a*]]*ς ∈ I then* [[Γ *▶* Δ]]*ς is valid.*

## Proof.

1. By Lemma [5.1](#_bookmark23) [[**at** *u*]]*ς* is not valid on any valuation *ς*. The result then follows from the definition of validity.
2. Since **[***a*]]*ς ∈ I* it follows that **[at** *a*]]*ς* is valid. By definition, either **[***G*]]*ς* is invalid for some *G ∈* Γ or **[***D*]]*ς* is valid for some *D ∈* Δ. Again by definition, [[Γ *▶* Δ]]*ς* .

**Theorem 5.7 (Soundness)** *Suppose* [[*-*]] *is a model. Then if* Γ *▶* Δ *is derivable then* [[Γ *▶* Δ]] *is valid.*

**Proof.** We work by induction on the depth of derivations.

* Most cases are immediate from the definitions (we have sketched the reasoning in what we consider the more complex cases in lemmas above). Only rule (**at L**) is not straightforward.
  + The case of (*∀***L**). By part (ii) of Lemma [5.3](#_bookmark25).
  + The case of (*∀***R**). By Lemma [5.5](#_bookmark27).
  + The case of (**at R**). By Lemma [5.1](#_bookmark23).
  + The case of (**at L**). Suppose Γ*,* **at** *a ▶* Δ is derivable where *a /∈* Γ*,* Δ. Let *a'* be such that **[***a'*]]*ς ∈ I*. By Lemma [4.1](#_bookmark19) for [*a*:=*a'*] it follows that Γ*,* **at** *a' ▶* Δ is derivable, and the derivation is no deeper. By inductive hypothesis **[**Γ*,* **at** *a' ▶* Δ]]*ς* is valid. The result now follows by part (ii) of Lemma [5.6](#_bookmark28).
  1. *Prime theories and completeness*

Let *T* range over (possibly infinite) sets of formulae; call *T* a ***theory***. Call *T* ***inconsistent*** when there exists some (finite) Γ *⊆ T* such that Γ *▶* is derivable. Otherwise call *T* ***consistent***. Call *T* ***maximal*** when:

* *T* is consistent.
* If Γ *⊆T* and Γ *▶ P* then *P ∈T* . We call *T* ***deductively closed***.
* For all *P* , *P ∈T* or *¬P ∈T* .
* If *P* [*a*:=*t*] *∈T* for all *t* then *∀a.P ∈T* .
* **at** *a ∈T* for at least one *a*.

**Lemma 5.8** *If T is maximal then ∀a.P ∈ T if and only if P* [*a*:=*t*] *∈ T for all t. Also ¬∀a.P ∈T if and only if ¬P* [*a*:=*t*] *∈T for some t.*

**Proof.** By easy calculations using the definition of maximal set.

**Lemma 5.9** *If* Γ *is a consistent and a /∈* Γ *then* Γ *∪ {***at** *a} is consistent.*

**Proof.** We suppose Γ*,* **at** *a ▶* and we prove a contradiction. By assumption *a /∈* Γ; it is not hard to extend the derivation of Γ*,* **at** *a ▶* to a derivation of Γ *▶ ∀a.¬***at** *a*.

We combine this using (**Cut**) with the following derivation

(**Ax**)

Γ*,* **at** *a ▶* **at** *a*

Γ*,* **at** *a, ¬***at** *a ▶*

(*⊃* **L**)

(*∀***L**)

Γ*,* **at** *a, ∀a.¬***at** *a ▶*

Γ*, ∀a.¬***at** *a ▶*

(**at L**)

to produce a derivation of Γ *▶*. This contradicts our assumption that Γ is consis- tent.

**Lemma 5.10** *If* Γ *is ﬁnite and consistent then there exists a maximal theory T*

*such that* Γ *⊆T .*

**Proof.** Since Γ is finite there are infinitely many variable symbols that do not appear in Γ. The language is countable so we enumerate its predicates *P*1*, P*2 *...* We now construct a sequence Γ0, Γ1, . . . as follows:

* Γ0 =Γ *∪ {***at** *b}* for some *b* such that *b /∈* Γ.
* Suppose Γ*n, Pn*+1 is inconsistent. Then Γ*n*+1 = Γ*n ∪ {¬Pn*+1*}*.
* Suppose Γ*n, Pn*+1 is consistent and *Pn* is not of the form *¬∀a.P* for any *P* . Then Γ*n*+1 = Γ*n ∪ {Pn*+1*}*.
* Suppose Γ*n, Pn*+1 is consistent and *Pn* is of the form *¬∀a.P* for some *P* . Then Γ*n*+1 = Γ*n ∪ {¬P* [*a*:=*b*]*}* for some *b /∈* Γ*n*.

Note that Γ*m ⊆* Γ*n* if *m ≤ n*. Write *T* = *n* Γ*n*. By construction Γ *⊆T* . If we can prove that *T* is maximal then we are done:

* *T* is consistent: Γ0 is consistent by Lemma [5.9](#_bookmark29). By construction Γ*n*+1 is consis- tent if Γ*n* is, so Γ*n* is consistent for all *n >* 0.

If *T* is inconsistent then it has a finite inconsistent subset. This must be contained in some Γ*n* and this is impossible. The result follows.

* *T* is deductively closed: Suppose that *T ▶ P* . *P ≡ Pn* for some *n* and by construction *P ∈* Γ*n*+1.
* For every *P* , either *P ∈ T* or *¬P ∈ T* : *P ≡ Pn* for some *n*. By construction

*Pn ∈* Γ*n*+1 or *¬Pn ∈* Γ*n*+1.

* If *P* [*a*:=*s*] *∈ T* for all *s* then *∀a.P ∈ T* . Suppose that *∀a.P /∈ T* , then *¬∀a.P ∈*

*T* . Now *¬∀a.P ≡ Pn* for some *n*. Therefore *¬P* [*a*:=*b*] *∈* Γ*n*+1 for some *b /∈* Γ*n* and so *P* [*a*:=*b*] */∈T* .

* **at** *b ∈T* for at least one *b*: By construction of Γ0.

For a given set of sentences *T* , define *|t|* = *{u | t* ~ *u ∈T }*.

**Lemma 5.11** *If T is maximal then |u|⊆ |t| if and only t* ~ *u ∈T .*

**Proof.** Suppose that *|u| ⊆ |t|*. Since *T ▶ u* ~ *u* it follows that *u ∈ |u|* and so if *|u| ⊆ |t|* we have that *u ∈ |t|* and so *t* ~ *u*. Conversely suppose *t* ~ *u ∈ T* . Now if *s ∈ |u|* then *u* ~ *s ∈ T* and so *t* ~ *s ∈ T* by the maximality of *T* and the transitivity of ~ (see Theorem [2.1](#_bookmark7)), and so *s ∈ |t|*. Thus, *|u|⊆ |t|*.

**Lemma 5.12** *If* Γ *is consistent then there exists some model* [[*-*]] *and some valuation*

*ς on* [[*-*]] *such that* [[Γ]]*ς is valid.*

**Proof.** Suppose Γ is consistent. Extend it using the construction in the proof of Lemma [5.10](#_bookmark30) to a maximal set *T* such that Γ *⊆T* .

Define an *a*-domain D by:

* *|*D*|* = *{|t|| t* is a term*}*
* *I|*D*|* = *{|t||* **at** *t ∈T }*
* *|t| ≤|*D*| |u|* when *|t|⊆ |u|*.

The arity **at** : (*↓*) and the rules for aequality entail that if **at** *t* and *t* ~ *u* are derivable then so is **at** *u*. It follows that *I* is down-closed. Therefore D is an *a*-domain.

We specify a model **[**- **]** and valuation *ς* by stipulating that:

* 1. If *f* has arity *n* and *|t*1*|,..., |tn|∈ |*D*|*. Then **[***f* ]](*|t*1*|,..., |tn|*)= *|f* (*t*1*,..., tn*)*|*.
  2. **[***p*]] = *{*(*|t*1*|,..., |tn|*) *| p*(*t*1*,..., tn*) *∈T }*.
  3. *ς*(*a*)= *|a|∈ |*D*|*.

We must verify the properties required of a model.

* If **[***f* ]](*|t*1*| ... |tn|*) *∈ I*, then *|f* (*t*1*,..., tn*)*| ∈ I* and so **at** *f* (*t*1*,..., tn*) *∈ T* . But this is impossible since **at** *f* (*t*1*,..., tn*) *▶* and *T* is consistent. Therefore [[*f* ]](*|t*1*| ... |tn|*) */∈ I*.

Also if *|ti| ≤ |t'|* then *|ti| ⊆ |t'|* and then *t'* ~ *ti ∈ T* . But since ~ has arity

*i*

(*↑, ↓*) we have that

*i* *i*

(~ **R**)

*▶ f* (*t*1*,..., t',..., tn*) ~ *f* (*t*1*,..., t',..., tn*)

*i i* (~ **L***↓*)

*t'* ~ *ti ▶ f* (*t*1*,..., t',..., tn*) ~ *f* (*t*1*,..., ti,..., tn*)

*i* *i*

and so *f* (*t*1*,..., t',..., tn*) ~ *f* (*t*1*,..., ti,..., tn*) *∈T* . Therefore

*i*

*|f* (*t*1*,..., ti,..., tn*)*|⊆ |f* (*t*1*,..., t',..., tn*)*|*

*i*

which entails that *|f* (*t*1*,..., ti,..., tn*)*| ≤ |f* (*t*1*,..., t',..., tn*)*|*. Thus if *|ti| ≤ |t'|*

*i* *i*

then **[***f* ]](*|t*1*|,..., |ti|,..., |tn|*) *≤* [[*f* ]](*|t*1*|,..., |t'|,..., |tn|*) and we have shown

*i*

that **[***f* **]** is monotone.

* By construction *ς* maps atoms to elements of D. By construction there exists an *a* such that **at** *a ∈ T* . Also by construction *|***at** *a| ∈ I*. Therefore there exists an *a* such that *ς*(*a*) *∈ I*.
* Suppose the arity of *p* is (*d*1 *... dn*) and suppose *|ti|≤ |t'|*. By definition *|ti|⊆ |t'|*

*i* *i*

and by Lemma [5.11](#_bookmark31) *t'* ~ *ti ∈T* . There are now three cases:

*i*

*·* The case that *di* = *↑*:

If (*|t*1*|,..., |ti|,..., |tn|*) *∈* [[*p* **]**, then *p*(*t*1 *..., ti,..., tn*) *∈ T* . By the following derivation and the maximality of *T* , we have that *p*(*t*1 *..., t',..., tn*) *∈T* :

*i*

(**Ax**)

*p*(*ts*)[*a*:=*ti*] *▶ p*(*ts*)[*a*:=*ti*] [*a↑p*(*ts*)] (~**L***↑*)

*t'* ~ *ti, p*(*ts*)[*a*:=*ti*] *▶ p*(*ts*)[*a*:=*t'*]

*i* *i*

Therefore (*|t*1*|,..., |t'|,..., |tn|*) *∈* [[*p*]].

*i*

*·* The cases *di* = *↓* and *di* = Ç are similar.

We prove by induction that **[***t*]]*ς* = *|t|* for any *t*:

* The case *t ≡ a*. **[***a*]]*ς* = *ς*(*a*)= *|a|* by construction.
* The case *t ≡ f* (*t*1*,..., tn*). Then

[[*f* ]]([[*t*1]]*ς ,...,* [[*tn*]]*ς* )= **[***f* ]](*|t*1*|,..., |tn|*)= *|f* (*t*1*,..., tn*)*|.*

We prove by induction that **[***P* ]]*ς* is valid if and only if *P ∈T* :

* The case that *P ≡* **at** *t*. **[at** *t*]]*ς* is valid when **[***t*]]*ς ∈ I*, which happens when

*|t|∈ I*, and this happens when **at** *t ∈T* .

* The case that *P ≡ t*1 ~ *t*2. **[***t*2 ~ *t*1]]*ς* is valid when **[***t*1]]*ς ≤* [[*t*2]]*ς* , which happens when *|t*1*| ≤ |t*2*|*, which happens when *|t*1*| ⊆ |t*2*|*, and by Lemma [5.11](#_bookmark31) this happens when (*t*2 ~ *t*1) *∈T* .
* The case that *P ≡ p*(*t*1*,..., tn*) for some *p /∈ {***at** *,* ~*}*. **[***p*(*t*1*,..., tn*)]]*ς* is valid

when ( **[***t*1]]*ς ,...,* [[*tn*]]*ς* ) *∈* [[*p*]]. Now [[*ti*]]*ς* = *|ti|* for 1 *≤ i ≤ n*, so this happens when (*|t*1*|,..., |tn|*) *∈* [[*p* **]**. By construction this happens when *p*(*t*1*,..., tn*) *∈T* .

* Other cases are straightforward. For example:

[[*∀a.P* ]]*ς* is valid iff **[***P* ]]*ς{a'→|t|}* is valid for all *|t|*

iff **[***P* [*a*:=*t*]]]*ς* is valid for all *t* (Lemma [5.2](#_bookmark24))

iff *P* [*a*:=*t*] *∈T* is valid for all *t* (induction hypothesis) iff *∀a.P ∈T*

We conclude that **[***P* ]]*ς* is valid for every *P ∈T* , and it follows that **[**Γ **]***ς* is valid.

**Theorem 5.13 (Completeness)** *If* [[Γ *▶* Δ]] *is valid in all models then* Γ *▶* Δ*.*

**Proof.** *a*-logic is classical. Therefore Γ *▶* Δ when Γ *∪ {¬D* : *D ∈* Δ*} ▶*. It suffices to show that if Γ *∪ {¬D* : *D ∈* Δ*}* is consistent then there is a model **[**- **]** and valuation *ς* such that **[***P* ]]*ς* is valid for all *P ∈* Γ *∪ {¬D* : *D ∈* Δ*}*. This follows by Lemma [5.12](#_bookmark32).

# Consistency and nontriviality of the axiomatisation of the *λ*-calculus

We build a model, in the sense of Subsection [5.1](#_bookmark22), of the theory for the untyped *λ*-calculus from Subsection [3.2](#_bookmark17). This illustrates an easy but non-trivial example of an *a*-logic model and it suffices to prove consistency of the axioms in Figures [2](#_bookmark14), [3](#_bookmark15) and [4](#_bookmark16).

Let Λ be the set of untyped *λ*-terms, generated by the grammar

*q, r* ::= *a | λa.q | qq.*

Here *a* ranges over variables of *a*-logic; it is convenient to identify these with vari- ables of the *λ*-calculus. We define *α*-conversion and *β*-reduction on *λ*-terms set as usual [[4](#_bookmark44)]. Write *fv* (*q*) for the free variables of *q*. For example *a /∈ fv* (*λa.a*).

Write *q*[*r/a*] for the capture-avoiding substitution of *r* for *a* in *q*, making some fixed but arbitrary choice about how to rename variable symbols to avoid capture. For example (*λa.b*)[*a/b*]= *λa'.a* for some fixed but arbitrary choice of *a'*.

**Remark 6.1** Note that our use of *q*[*r/a*] for capture-avoiding substitution in *λ*-

terms is distinct from the (syntactic) substitution action [*a*:=*t*] of Subsection [2.2](#_bookmark0): [*q/a*] is an operation on the *λ*-terms which we are about to use to construct a concrete model for the theory of Figures [2](#_bookmark14) and [4](#_bookmark16); [*a*:=*t*] is an operation on *a*-logic terms.

Write *q →αβ q'* for the relation generated by an *α*-conversion followed by a

*β*-reduction followed by *α*-conversion, and call this *αβ****-reduction***. [6](#_bookmark35)

Write *→∗*

*αβ*

for the transitive reflexive closure of *→αβ*. That is, *q' →∗*

*q* when

either *q'* = *q* or there exists some chain *q*2*,..., qn−*1 such that

*αβ*

*q' →αβ q*2 *→αβ ... →αβ qn−*1 *→αβ q.*

Write *q'↓αβ* for the relational image of *{q'}* under *→∗*

*αβ*

. That is,

*q'↓αβ* = *{q | q' →∗*

*αβ*

*q}.*

All of these definitions are standard [[4](#_bookmark44)].

**Definition 6.2** Let ¿ be some symbol not in Λ and read it as ***error***. We define an

*a*-domain D by:

* *|*D*|* = *{q↓αβ | q ∈* Λ*}∪ {*¿*}*. *x, y, z* will range over elements of *|*D*|*.
* *x ≤ y* when *x ⊆ y* or *x* = *y* = ¿.
* *I* = *{{a}| a ∈* A*} ∪ {∅}*.

We define a valuation *ς* by *ς*(*a*)= *{a}*. We define a model **[**- **]** by:

* [[*λ*]](*x, y*)= (*λa.r*)*↓αβ* if *x ∩* A = *{a}* and *y* = *r↓αβ* .

[[*λ*]](*x, y*)= *{*¿*}* otherwise.

* *x*[[ *·* ]]*y* = (*qr*)*↓αβ* if *x* = *q↓αβ* and *y* = *r↓αβ* .

*x*[[ *·* ]]*y* = *{*¿*}* otherwise.

* [[*⟨⟩*]](*z, x, y*)= (*q*[*r/a*])*↓αβ* if *z* = *q↓αβ* , *x ∩* A = *{a}*, and *y* = *r↓αβ* .

[[*⟨⟩*]](*z, x, y*)= *{*¿*}* otherwise.

* (*x, y*) *∈* **[**# **]** when *x ∩* A = *{a}* and *y* = *r↓αβ* and *a /∈ fv* (*r*). (*x, y*) */∈* **[**# **]** otherwise.

*λ*, *·*, and *⟨⟩* in the language of *L* are term-formers with arities 2, 2, and 3 respectively. *a*-logic permits ‘silly terms’ such as *λ*(*λa.λb.ab*)*.c*. Standard methods are available to exclude them; for instance a sorting system for *a*-logic could be developed. On the other hand, we could even ‘embrace the silliness’, allow such terms, and investigate their properties. All of this is future work; for our purposes we need only include ¿ an error value, and in Definition [6.2](#_bookmark34) we let the denotation of a silly term such as *λ*(*λa.λb.ab*)*.c* be *{*¿*}*. For more future work, it may be possible to identify ¿ with the *⊥* of Scott domain models of the *λ*-calculus [[27](#_bookmark67)]. However,

6 The first author has studied different ways these definitions can be expressed. This is not the issue for constructing the model; for our purposes it suffices to know that the relation exists.

this requires a more sophisticated construction, perhaps involving nominal domain theory [[26](#_bookmark66)].

We now sketch a proof that we have indeed constructed a model of *L*. Lem- mas [6.3](#_bookmark36) and [6.4](#_bookmark37) are technical lemmas needed for Corollary [6.5](#_bookmark38).

**Lemma 6.3** *Suppose x, x' ∈ |*D*| and suppose x ≤ x'. Then:*

* *If x'* = *r'↓αβ for some r' then x* = *r↓αβ for some r ∈ x' (that is, x* = *r↓αβ for*

*some r such that r' →∗ r).*

*αβ*

* *If x' ∩* A = *{a} then x ∩* A = *{a}.*

## Proof.

* Suppose *x'* = *r'↓αβ* and suppose *x ≤ x'*. Note that ¿ */∈ x'*. By definition of *≤* we know *x ⊆ x'*. It follows that ¿ */∈ x* and so *x* = *r↓αβ* for some *r*. It follows that

*r ∈ x'* and therefore that *r' →∗ r*.

*αβ*

* Suppose *x' ∩* A = *{a}* and *x ≤ x'*. By construction of *|*D*|* it follows that *x'* = *r'↓αβ*

and *r' →∗ a*. By the first part of this result *x* = *r↓αβ* and *r' →∗ r*. It is a fact

*αβ*

*αβ*

that *a* does not *β*-reduce. By confluence of *λ*-calculus reduction it follows that

*∗ a*. The result follows.

*r →*

*αβ*

**Lemma 6.4** *If x ∈ |*D*| then precisely one of the following holds:*

* *x ∩* A = *{a} for some a ∈* A*.*
* *x ∩* A = *∅.*

**Proof.** We work by cases on possible forms for *x ∈ |*D*|*:

* The case *x* = *q↓αβ* . It is a fact of *λ*-calculus reduction that if *q →∗*

*αβ*

*a* for some

*a ∈* A then *q /→∗ b* for any other *b*.

*αβ*

* The case *x* = *{*¿*}*. Immediate.

**Corollary 6.5** [[*λ*]]*,* [[ *·* ]]*, and* [[*⟨⟩*]] *are monotone.*

**Proof.** It suffices to verify that:

* If *x ≤ x'* and *y ≤ y'* then **[***λ*]](*x, y*) *≤* [[*λ*]](*x', y'*).
* If *x ≤ x'* and *y ≤ y'* then **[** *·* ]](*x, y*) *≤* [[ *·* ]](*x', y'*).
* If *x ≤ x'*, *y ≤ y'*, and *z ≤ z'* then **[***⟨⟩*]](*x, y, z*) *≤* [[*⟨⟩*]](*x', y', z'*).

We will consider only the first of these calculations, for *λ*; the calculations for *·*

and *⟨⟩* are similar. Suppose that *x ≤ x'* and *y ≤ y'*.

By Lemma [6.4](#_bookmark37) either *x' ∩* A = *∅* or *x ∩* A = *{a}* for some *a ∈* A. It is helpful, though, to distinguish three cases:

* The case that *y'* = *{*¿*}*. Then **[***λ*]](*x', y'*) = *{*¿*}*. We assumed *y ≤ y'*. By construction of *|*D*|* it follows that *y* = *{*¿*}*. Therefore **[***λ*]](*x, y*) = *{*¿*}* and the result follows.
* The case that *x' ∩* A = *∅*. Then **[***λ*]](*x', y'*)= *{*¿*}*. We assumed *x ≤ x'* and so (since the definition of *≤* unpacks to *x ⊆ x'*) *x∩* A = *∅*. Therefore **[***λ*]](*x, y*)= *{*¿*}* and the result follows.
* The case that *x' ∩* A = *{a}* and *y'* = *r'↓αβ* . Then **[***λ*]](*x', y'*)= (*λa.r'*)*↓αβ* . By part 1 of Lemma [6.3](#_bookmark36) since *x ≤ x'* it must be that *x ∩* A = *{a}*. By part 2 of

Lemma [6.3](#_bookmark36) since *y ≤ y'* it must be that *y* = *r↓αβ* for some *r* such that *r' →∗ r*.

*αβ*

By construction **[***λ*]](*x, y*)= (*λa.r*)*↓αβ* . By properties of *αβ*-reduction *λa.r' →∗ λa.r* and so ( **[***λ*]](*x, y*) *⊆* [[*λ*]](*x', y'*) and so by definition) **[***λ*]](*x, y*) *≤* [[*λ*]](*x', y'*) as required.

*αβ*

**Theorem 6.6** D *is an a-domain. ς is a valuation.* [[*-*]] *is a model.*

**Proof.** We verify all the required properties in turn. D is an *a*-domain:

* *|*D*|* is a set: This is a fact.
* *⊆* is a partial order on *|*D*|*: This is also a fact.
* *I* is down-closed: Suppose *x' ∈ I* and *x ≤ x'*. By definition of *I* we know that *x'* = *{a}* for some *a ∈* A. By part 1 of Lemma [6.3](#_bookmark36) we know *x* = *x'*. It follows that *x ∈ I*.

*ς* is a valuation: By construction *ς* maps *a* to *{a} ∈ |*D*|*. Also, *ς*(*a*) *∈ I* for all

*a ∈* A, and therefore *ς*(*a*) *∈ I* for at least one *a ∈* A. **[**- **]** is a model:

* *λ* is a term-former with arity 2. By construction **[***λ* **]** is a function from *|*D*|*2 to

*|*D*|*. It is a fact of *λ*-calculus reduction that if *λa.q →∗ q'* then *q'* is not a variable

*αβ*

symbol. It follows that **[***λ*]](*x, y*) */∈ I* always. We proved in Corollary [6.5](#_bookmark38) that [[*λ* **]** is monotone.

* The cases of application and substitution are similar.
* Recall that # : (*↓, ↓*). Suppose that *x ≤ x'*, *y ≤ y'*, and (*x', y'*) *∈* [[#]]. We must show that (*x, y*) *∈* [[#]].

By construction *x'* = *{a}* for some *a ∈* A, *y'* = *r'↓αβ* , and *a /∈ fv* (*r'*). By part 1 of Lemma [6.3](#_bookmark36) since *x ≤ x'* we know *x* = *{a}*. By part 2 of Lemma [6.3](#_bookmark36) since

*y ≤ y'* it must be that *y* = *r↓αβ* for some *r* such that *r' →∗ r*. It is a fact of the

*αβ*

*λ*-calculus that *a /∈ fv* (*r*). The result follows.

**Theorem 6.7** *The construction above yields a model of L.*

**Proof.** We verify that the axioms given in in turn. We consider only two cases. (*'→*#)

We must show that if (*x, y*) *∈* **[**# **]** then

* + *x* = *{a}∈ I*,
  + for all *{b}∈ I* such that *b /*= *a* it is the case that *y ⊆* [[*⟨⟩*]](*y, {a}, {b}*).

Suppose that (*x, y*) *∈* **[**# **]**. By definition *x* = *{a}*, so *x ∈ I*. Also by definition

*y* = *r↓αβ* and *a /∈ fv* (*r*). By definition **[***⟨⟩*]](*r↓αβ , {a}, q↓αβ* ) = (*r*[*q/a*])*↓αβ* .

But, since *a /∈ fv* (*r*), *r*[*q/a*] *≡ r*. And so *y* = *r↓αβ ⊆ r*[*q/a*])*↓αβ* . The result follows.

(*'→***comm**)

Suppose that *x*2 = *{a*2*}∈ I*, (*x*1*, x*2) *∈* **[**# **]** and (*x*1*, y*2) *∈* **[**# **]**. We must show

that

* [[*⟨⟩*]]

[[*⟨⟩*]](*z, x*2*, y*2)*, x*1*,* [[*⟨⟩*]](*y*1*, x*2*, y*2)

*⊆* [[*⟨⟩*]]

[[*⟨⟩*]](*z, x*1*, y*1)*, x*2*, y*2

We may suppose that *y*1 = *r*1*↓αβ* , *y*2 = *r*2*↓αβ* and *z* = *q↓αβ* . It follows that *x*1 = *{a*1*} ∈ I* and that *a /∈ fv* (*r*2) . It also follows that *a*1 */∈ fv* (*a*2) and, since *a*2 is just a variable, this entails that *a*1 */*= *a*2.

Now **[***⟨⟩*]](*z, x*2*, y*2)= *q*[*r*2*/a*2]*↓αβ* and **[***⟨⟩*]](*y*1*, x*2*, y*2)= *r*1[*r*2*/a*2]*↓αβ* by defi- nition. So **[***⟨⟩*]](*q*[*r*2*/a*2]*↓αβ, x*1*, r*1[*r*2*/a*2]*↓αβ* )= *q r*2*/a*2 *r*1[*r*2*/a*2]*/a*1 *↓αβ* .

Similarly it follows that **[***⟨⟩*]] [[*⟨⟩*]](*z, x*1*, y*1)*, x*2*, y*2 = *q*[*r*1*/a*1][*r*2*/a*2]*↓αβ*

But it is a fact of the *λ*-calculus that if *a*1 */*= *a*2 and *a /∈ fv* (*r*2) that

*q*[*r*1*/a*1][*r*2*/a*2] *≡ q* *r*2*/a*2 *r*1[*r*2*/a*2]*/a*1 .

The other axioms in Figures [2](#_bookmark14), [3](#_bookmark15) and [4](#_bookmark16) can be verified similarly.

# Conclusions

*a*-logic is a classical first-order logic with a two-valued model theory, one of many [[5](#_bookmark45),[13](#_bookmark53)]. The twist to the story is that **at** can identify variables, and accordingly truth-values are given also to open predicates.

In the semantics, predicates are sets of (tuples of) elements of the domain. **at** is a unary predicate. Atoms are simply a subset of the underlying domain; those in [[**at ]** which is the interpretation of **at** in the domain. Elements of **[at ]** are special; they cannot be in the image of the interpretation of any term-formers.

We believe that *a*-logic could provide a logical semantics for *isvar* in PROLOG, as used for example in practical programming in CIAO PROLOG [[9](#_bookmark49)]. *isvar* is a *non-logical predicate* which is true when its argument has not yet been unified with a concrete value — that is, when it is a variable. This is used exactly as we have used it in this paper, to master shallow embeddings into CIAO of languages with variables. **at** and *isvar* appear to be compatible but problems arise interpreting PROLOG conjunction, which ceases to be commutative in the presence of *isvar* . This is a known issue which we are attempting to solve by using a modified translation of CIAO conjunction which is will be the topic of a future paper.

A previous paper considered a variant of *a*-logic with equality [[15](#_bookmark55)]. There, in order to axiomatise substitution we had to introduce a (in the authors’ opinion) *ad hoc* notion which we called ‘essentially a term’. In this paper we take a different path and introduce a *directed* equality ~ — the *arrow* in the title of this paper. This solves the issues we encountered in [[15](#_bookmark55)], arguably in more elegant way which seems to benefit the resulting theory and axioms. In particular, our principal axiomatisations (of the *λ*-calculus and of substitution) are improved.

It is relevant how conveniently *a*-logic can be implemented in a theorem-prover such as Isabelle [[21](#_bookmark60)]. We do not know.

~ gives *a*-logic a flavour of a ‘logical theory of rewriting’. [7](#_bookmark40) Rewriting itself is traditionally intensional on the syntax of terms (we look at a term’s syntax to decide whether to rewrite), or completely abstract (we just take any relation on a set, but the only properties of its elements are the relation so elements are closed, in the sense that instantiation is absent) [[3](#_bookmark43),[28](#_bookmark68)]. In some sense **at** provides just enough to give a point of contact between these two worlds and it may be possible to make this idea useful for the theory of rewriting.

How else can the *λ*-calculus (and friends) be modelled in a formal system and how does *a*-logic relate to them?

We can use a higher-order system to begin with (build the system as an extension of a *λ*-calculus). The *λ*-calculus cannot recognise a variable symbol as such, so to use it to model the *λ*-calculus requires deep embedding which model variables by closed terms (in essence, you have to write a Turing machine).

First-order logic also cannot recognise a variable symbol. We can take a two-level hierarchy with object-level variables modelled by constants. We cannot quantify over constants so a hallmark of such approaches is an infinity of axioms, one for each constant (the *a*-logic axiomatisations look similar, but are finite). This approach is followed for example by Cylindric Algebras [[8](#_bookmark48)] (which may assume only finitely many variable symbols so as to obtain a finite theory), by Salibra’s Lambda Abstraction Algebras [[24](#_bookmark64)], and by some treatments of Structural Operational Semantics (SOS) [[1](#_bookmark41)], see for example Bernstein’s SOS axiomatisations of the *π*-calculus, the lazy *λ*-calculus, and CHOCS [[6](#_bookmark46)].

We can do away with variables entirely, for example using Boehm trees, Scott domains, or combinators [[4](#_bookmark44)]. The translations are only for closed terms (hence, not compositional) and combinators may suffer exponential blow-up [[7](#_bookmark47)]. This is a bit off-topic for us, because we are interested not just in a ‘naked denotation’ but also in the formal system it is constructed in.

We can suppose a two-level hierarchy of variables and meta-variables and as- sume *β*-reduction for the former; computation/reasoning is by instantiation of meta- variables following by reduction. One of several examples of this style is Combina- tory Reduction Systems (CRS). Note that in CRS meta-variables vary over closed elements — the higher-order elements are used to feed arguments to where they are used. In *a*-logic unknowns may be substituted for open terms. So for example *a*#*b* states that (whatever *b* is) it should not depend on *a*. This is not expressible in a CRS. Being a variable or meta-variable is an intensional property in CRS, whereas in *a*-logic the difference is expressed by **at** and can be mixed with implication and the rest of the logic. Finally, in CRS *β*-reduction is built-in. Although we certainly concentrate on substitutions and the *λ*-calculus in this paper, other theories are possible so *a*-logic is probably more flexible.

Nominal logic [[22](#_bookmark59)] has a notion of atom but keeps this separate from the notion of variable; this is because nominal logic was designed for reasoning *on* syntax — in

7 To our surprise, we could find no reference in the literature for a logic-based-on a transitive reflexive not-necessarily-symmetric congruence — that is, for a logic of rewriting. We would be happy to hear of such a reference. Rewriting Logic [[19](#_bookmark61)] seems different, informally treating terms as formulae.

our terminology, reasoning on a deep embedding where there is a clear distinction between variable symbols of the object-level language, called atoms because they behave like atomic entities, and variables of the meta-level language which range over unknown elements of the domain. A shallow embedding of (for example) the *λ*-calculus in nominal logic is not possible. We also believe, but have not proved, that *a*-logic is consistent with arbitrary choice. Nominal logic is not [[22](#_bookmark59)], so *a*-logic may have some interesting technical advantages.

*a*-logic may be closer to Miller and Tiu’s logic of Generic Judgements (GJ) [[20](#_bookmark62)], where ‘being a variable symbol’ is managed by scoping for a dedicated quantifier

*∇* instead of a dedicated predicate **at** . GJ is significantly different in being higher- order and having a ‘definitional’ equality [[25](#_bookmark65)] suited more to logic-programming than model theory; we suspect it is unsuited for our applications.

Kit Fine’s ‘Reasoning with arbitrary objects’ [[12](#_bookmark52)] axiomatises dependence be- tween elements. Perhaps our semantics, which contain variables, may be related.

# References

1. L. Aceto, W. Fokkink, and C. Verhoef. Structural operational semantics. In *Handbook of Process Algebra*. Elsevier, 1999.
2. A. Azurat and I.S.W.B. Prasetya. A survey on embedding programming logics in a theorem prover. Technical Report 7, Institute of information and computing sciences, Utrecht University, 2002.
3. Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, Great Britain, 1998.
4. H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics (revised ed.)*. North-Holland, 1984.
5. John Bell and Mosh´e Machover. *A course in mathematical logic*. North-Holland, 1977.
6. Karen L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In *Logic in Computer Science*, pages 153–164, 1998.
7. Martin W. Bunder. Some improvements to Turner’s algorithm for bracket abstraction. *J. Symb. Log.*, 55(2):656–669, 1990.
8. S. Burris and H. Sankappanavar. *A Course in Universal Algebra*. Springer, 1981.
9. CIAO. CIAO Prolog.

[www.clip.dia.fi.upm.es/Software/Ciao/ciao](http://www.clip.dia.fi.upm.es/Software/Ciao/ciao) html/ciao toc.html.

1. Marcel Crabb´e. On the notion of substitution. *Logic Journal of the IGPL*, 12(2):111–124, 2004.
2. Maribel Fern´andez and Murdoch J. Gabbay. Nominal rewriting. *Information and Computation*, 205:917–965, 2007.
3. Kit Fine. *Reasoning with Arbitrary Objects*. Blackwell, 1985.
4. Dov Gabbay and Franz Gu¨nthner, editors. *Handbook of philosophical logic*. Number 166 in Synthese Library. D.Reidel Publishing company, 1986.
5. Murdoch J. Gabbay and J. Cheney. A sequent calculus for nominal logic. In *Proc. 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 139–148. IEEE Computer Society, 2004.
6. Murdoch J. Gabbay and Michael J. Gabbay. a-logic. In *We Will Show Them: Essays in Honour of Dov Gabbay*, volume 1. College Publications, 2005.
7. Murdoch J. Gabbay and Aad Mathijssen. Capture-avoiding substitution as a nominal algebra. In *ICTAC’2006: 3rd International Colloquium on Theoretical Aspects of Computing*, volume 4281 of *LNCS*, pages 198–212, 2006.
8. Murdoch J. Gabbay and Aad Mathijssen. Capture-avoiding substitution as a nominal algebra. *Formal Aspects of Computing*, 2008. Available online.
9. Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding.

*Formal Aspects of Computing*, 13(3–5):341–363, 2002.

1. Narciso Mart´ı-Oliet and Jos´e Meseguer. Rewriting logic: roadmap and bibliography. *Theor. Comput.* *Sci.*, 285(2):121–154, 2002.
2. Dale Miller and Alwen Tiu. A proof theory for generic judgments: An extended abstract. In *Proc. of LICS 2003*, pages 118–127. IEEE, 2003.
3. Larry Paulson. *The Isabelle reference manual*. Cambridge University Computer Laboratory, 2001.
4. Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186(2):165–193, 2003.
5. Antonino Salibra. On the algebraic models of lambda calculus. *Theoretical Computer Science*, 249(1):197–240, 2000.
6. Antonino Salibra. Lambda calculus: models and theories. In *AMiLP-2003*, number 21 in TWLT Proceedings, pages 39–54, 2003.
7. Peter Schroeder-Heister. Definitional reflection and the completion. In *Extensions of Logic Programming*, volume 798 of *Springer Lecture Notes in Artificial Intelligence*, pages 333–347, 1993.
8. Mark R. Shinwell and Andrew M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342(1):28–55, 2005.
9. Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, Cambridge, MA, USA, 1977.
10. Vincent van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, 1994.
11. Martin Wildmoser and Tobias Nipkow. Certifying machine code safety: Shallow versus deep embedding. In *TPHOLs*, volume 3223 of *LNCS*, pages 305–320. Springer, 2004.