How to prove theorems using dialogue

Hugo Fuks, Marcelino C. Pequeno & Martin Sadler
Department of Computing,
Imperial College of Science and Technology,
180 Queen's Gate, London UK, SW7 2BZ.
Telephone: 01 589-5111 ex 5064
Email: hf@doc.ic.ac.uk

Abstract

The proof of a theorem can be seen as a dialogue between nature and the logician who wants to prove it. Although this is not a new idea, there are no real examples of automatic theorem provers that are based on this way of seeing proof activity. This paper introduces a program/action logic presentation of a dialogue system first introduced by Hamblin [Ham71] and extended by Mackenzie [Mac79a]. An automatic dialogic theorem prover for propositional calculi is outlined.

0 Introduction

The act of proving a theorem can be seen as a dialogue. This statement can be fleshed out, as in the work of Hintikka [Hin73], by treating such a dialogue as a game between nature and the logician. Proof amounts to the existence of a winning strategy for the game.

The work this paper reports is based on the approach to dialogue logic created by Hamblin [Ham71] and extended by Mackenzie [Mac79a]. We have recast their presentations of systems into a program (or action) logic notation [Gol82], which is more amenable for establishing an appropriate metatheory and is notationally more succinct.

We see such an approach to logic as being the appropriate one on which to base a formal model of the software development process [Fin87]. In particular we are concerned with modelling the passing of information between members of a software team in the form of contracts (agreements or commitments to perform certain tasks).

Within the context of a project support environment based on such a model, one tool we are interested in providing is a dialogic framework for theorem proving. Furthermore we want the users of the environment to be able to see the theorem provers in the "same way" they see other dialogical activity.

Proceedings of the 5th Israeli Symposium on Artificial Intelligence,
Information Processing Association of Israel, December 1988.

1 Outline

This report consists of two parts: the introduction of the dialogue rules of System DC [Mac81] and the outlining of a dialogic framework for theorem proving.

In the first part we will give a brief account of System DC, in order to establish a new framework dialogue rules using an action logic presentation. The second part of the report is also subdivided into two parts. The first consists of the tailoring of these rules for the specifically use of the theorem prover. In the last part, we outline a recasting of an automatic theorem prover for propositional calculus due to Gabbay [Gab84] in this new setting.

2 System DC

Hamblin in his paper "Mathematical models of dialogue" [Ham71], proposed a formalism for the analysis of dialogues in terms of commitment sets, rules and syntactic relations. Mackenzie improved this formalism with systems DT [Mac79a], DC & DD [Mac79b] and DC+ [Mac81]. The focus of these attempts (see also [Mac80, Mac84]), has been to work out a solution for a major weakness of Hamblin's original system namely: 'begging the question' in a dialogue. From the point of view of an automated system this amounts to preventing a certain kind of looping in the search for a proof.

In this system, a dialogue is an ordered set of locution events, each represented by a triple of the form <stage, participant, locution-act>. There are locution modifiers that can be seen as functions applied to statements forming the locution-acts. They are **I** for identity; **Q** for question; **W** for withdrawal; **Y** for challenge; **G** for ground; **R** for resolution demand and **D** for denial.

Each participant has its own commitment store (**Cs**) - a book-keeping of its part of the dialogue - that is filled with locution-acts and statements during the course of the dialogue. The updating of these commitment stores is done by a set of Commitment Rules. For example:

Commitment Rule Identity:

After the assertion of a statement <stage_n, participant, **I**(s)>, {s} is added to both commitment stores.

Commitment is different from belief (or knowledge) in the sense that it doesn't bring up the epistemological complexity of the latter and it is always kept public.

Various syntactic relations are required to set up the rules. In order to define these rules, it

is necessary to have at least the schema for modus ponens to define the logic that governs the dialogue. The first relation is the modus ponens consequence **Mpc**. Given that T stands for a set of statements, T **Mpc** {s} means that using modus ponens only, {s} is immediately derivable from T. For example, if T is {t, t→s} then {s} is a modus ponens consequence. **Imc**, immediate consequence is the general case when any schema is used the same way it was presented for **Mpc**, inclusive for the modus ponens itself.

The second relation is that of immediate consequence conditional **ICC**. Basically is the case when in one locution-act, there is a conditional that is the instantiation of the modus ponens schema. For example if one of the participants had uttered the locution-event <stage_n, participant, **I**(if t and t→s then s)>, the locution-act (if t and t→s then s) is an **ICC**. Generally speaking, it is an instantiation of any of the schematta that define the logic that governs the dialogue in question.

The third relation is that of immediate inconsistency between statements **IMN**. A set of statements S is immediately inconsistent if it consists either of a statement together with its denial, or some finite set Z which is a sub-set of S together with the denial of an immediate consequence of Z.

Acc for acceptability is a complex syntactic relation, loosely: a statement {s} is acceptable to the commitment store (A:) just in case if either A is not commited at stage n to the challenge of {s}, or there is a set T of statements from (A:) to each of which A is commited at n, none of which are challenged at n, and {s} follows by modus ponens from T.

The last syntactic relation is the one that defines which are the allowable answers to a question. Given that a participant questions a statement, the only allowable answers are either the assertion of the same statement or its denial or its withdrawal.

Finally there is a set of Dialogue Rules formulated in term of commitments, locution-acts and the various syntactic relations that ensures that the dialogue will remain legal. For example,

Dialogue Rule Question:

After a question <stage_n, A, **Q**(s)> the next step must be either the assertion of the statement <stage_n+1, B, **I**(S)> or its withdrawal <stage_n+1, B, **W**(s)> or its denial <stage_n+1, B, **D**(s)>.

A careful description of both sets of rules is given in [Mac81].

3 An action logic presentation

Originally the rules of System DC are divided into Commitment Rules and Dialogue Rules. This division seems to be very useful to describe or analyse dialogues. Unfortunately it does not fit properly in order to prescribe dialogues. It is more convenient to see both these sets of rules in a more "Horn-clause like way".

The new set of rules combines the Commitment Rules with the Dialogue Rules. Each rule of this new set sees the actions of a dialogue in terms of state transformations. These rules essentially have the format:

Pre-condition $\Rightarrow$ [Agent , Action] Post-condition

This has the informal reading : "if Agent performs Action and it terminates, then given Pre-condition is true prior to Action, Post-condition is true after Action".

An important variation is:

Pre-condition $\Rightarrow$ [Agent-B, Action-of-B][Agent-A, Action-of-A] Post-condition

This has the informal reading : "if Agent-B performs Action-of-B and it terminates, then given Pre-condition is true prior to Action-of-B and if Agent-A performs Action-of-A and it terminates, Pos-condition is true after Action-of-A".

The pre- and post- conditions will take the form:

(Agent-B: set of commitments)(Agent-A: set of commitments)

and we will use 'b', 'a' as metavariables over such sets of commitments and '+', '-' for the operations of adding/deleting (if present) commitments to/from the sets.

4 The Dialogue Rules

The Dialogues Rules shown below, are the combination of the Commitment Rules with the Dialogue Rules of System DC using the Action Logic presentation of the previous section.

1) Rule Identity (Statements)

(B:b) (A:a) $\Rightarrow$ [Agent, I(s)] (B:b + {s}) (A:a + {s})

After Agent has asserted {s}, {s} is added to both commitment stores.

2) Rule Quest (Question)

$$(B{:}b)\ (A{:}a) \Rightarrow [B, \mathbf{Q}(s)][A, \mathbf{D}(s)]\ (B{:}b + \{\neg s\})\ (A{:}a + \{\neg s\}) \quad \text{or}$$
$$[A, \mathbf{I}(s)]\ (B{:}b + \{s\})\ (A{:}a + \{s\}) \quad \text{or}$$
$$[A, \mathbf{W}(s)]\ (B{:}b)\ (A{:}a - \{s\}))$$

After B has questioned {s}, both commitment stores remain the same. Given that, A denies {s} or asserts {s} or withdraws {s}. In the first case {¬s} is added to both commitment stores. In the second case {s} is added to both commitment stores. Finally in the third case {s} is removed from (A:) and (B:) remains the same.

3) Rule Ground (Answer to a Challenge)

$$(B{:}b\ \mathbf{Acc}\ \{t, t \rightarrow s\})\ (A{:}a)$$
$$\Rightarrow [B, \mathbf{Y}(s)][A, \mathbf{G}(t)]\ (B{:}b - \{s\} + \{\mathbf{Y}(s), t \rightarrow s, t\})\ (A{:}a + \{s, t \rightarrow s, t\})$$

After B has challenged {s}, {s} is removed from (B:) and {**Y**(s)} is added to it; {s} is added to (A:). Given that, if {t} is an acceptable answer to the challenge and (B:) accepts {t→s, t}, A answers {t} and then {t→s, t} are added to both commitment stores.

4) Rule Chall-W (Withdrawal after Challenge)

$$(B{:}b)\ (A{:}a) \Rightarrow [B, \mathbf{Y}(s)][A, \mathbf{W}(s)]\ (B{:}b - \{s\} + \{\mathbf{Y}(s)\})\ (A{:}a - \{s\})$$

After B has challenged {s}, {s} is removed from (B:) and {**Y**(s)} is added to it and {s} is added to (A:). Given that, A withdraws {s} from its commitment store and then {s} is removed from (A:) and (B:) remains the same.

5) Rule LogChall (Logical Challenges)

$$(B{:}b)\ (A{:}a)\ (s \in \mathbf{ICC}) \Rightarrow [B, \mathbf{Y}(s)][A, \mathbf{R}(s)]\ (B{:}b - \{s\} + \{\mathbf{Y}(s)\})\ (A{:}a + \{s\})$$

After B has challenged {s} and being {s} an immediate consequence conditional of (B:), {s} is removed from (B:) and {**Y**(s)} is added to it and {s} is added to (A:). Given that, A asks for a resolution demand of {s} and then both commitment stores remain the same.

6) Rule Resolve-Y (Resolve after Challenge)

(B:b)(A:a)(Tb **Imc** {s}) ⇒ [B, **Y**(s)] [A, **R**(Tb/s)] (B:b - {s}+{**Y**(s)}) (A:a+ {s})

After B has challenged {s} and being {s} an immediate consequence of Tb - the subset of (B:) from which {s} is immediately derivable - {s} is removed from (B:) and {**Y**(s)} is added to it and {s} is added to (A:). Given that, A asks for a resolution demand of Tb and then both commitment stores remain the same.

7) Rule Resolve-W (Resolve after Withdrawal)

(B:b) (A:a) (Tb **Imc** {s}) ⇒ [B, **W**(s)] [A, **R**(Tb/s)] (B:b - {s}) (A:a)

After B has withdrawn {s} and being {s} an immediate consequence of Tb - the subset of (B:) from which {s} is immediately derivable -, {s} is removed from (B:) and (A:) remains the same. Given that, A asks for a resolution demand of Tb and then both commitment stores remain the same.

8) Rule Resolution-W (Withdrawal after Resolution Demand)

(B:b) (A:a) (**IMN** A:) ⇒ [B, **R**(Ta)] [A, **W**(s)] (B:b) (A:a - {s})

Given that (A:) is immediately inconsistent, B asks for a resolution demand of Ta - the subset of (A:) that is immediately inconsistent - both commitment stores remain the same. A withdraws {s} thus restoring consistency to its commitment store and then {s} is removed from (A:) and (B:) remains the same.

9) Rule Resolution-I (Statement after Resolution)

(B:b) (A:a) ⇒ [B, **R**(Ta/s)] [A, **I**(s)] (B:b + {s}) (A:a + {s})

Given that {s} is an immediate consequence of Ta - the subset of (A:) from which {s} is immediately derivable -, and A refuses to admit it, B asks for a resolution demand of Ta with both commitment stores remaining the same. A asserts {s} thus restoring consistency to its commitment store and then {s} is added to both commitment stores.

5 A dialogic framework for theorem proving

As was said in the introduction of this report, the proof of a theorem can be seen as a dialogue. But rather than the 'logician' and 'nature' we would like to see the two participants of this dialogue as the 'heuristic searcher' and the 'verifier'.

The axioms of the theory from which the theorems should be proved build up the Database. The Logic Warehouse is the framework within which the logic (that governs the dialogue) is defined. It consists of inferences rules and axioms schemata. The proof starts with the challenge of the original theorem, and go on with successives challenges to the sentences added to the challenger's commitment store liable to a challenge -in fact, in the theorem prover we present, only the given answers can be challenged- and finishes when no sentences are liable to a challenge. One sentence is not liable to a challenge, if it belongs to the Database - is an axiom of the theory -, or is an instantiation of one axiom schemata of the Logical Warehouse -is a logical axiom.

The sentences introduced in the commitment stores after an answer, immediately derives - through the application of one inference rule- the sentence being challenged. Therefore when the dialogue is finished, all sentences in the Answer's commitment store are either axioms of the theory, or logical axioms, or are derived from a subset of the commitment store by an inference rule.

6 The Theorem Prover

The Automatic Dialogic Theorem Prover that is briefly outlined through the rest of this report is based on the one presented in [Gab84]. The main difference is of course its dialogical mode of operation.

We present an automatic theorem prover for the propositional classical logic and the fragment of intuitionistic logic with the connectives '$\wedge$' (and), '$\rightarrow$ ' (implication) and ' $\neg$ ' (not) (we do not deal with the intuitionistic disjunction, neither its classical translation is intuitionistically valid). The disjunctive formula A / B is translated as $\neg A \rightarrow B$ .The formulas are written in a specific form called D-formula. A D-formula is inductively defined as:

i) If A is a literal (a positive or negative atom) then A is a D-formula;

ii) If A1, A2 are D-formulas then $A1 \wedge A2$ is a D-formula;

iii) If A1, A2, ... ,An are D-formulas and B is a literal then $A1 \wedge A2 \wedge ... \wedge An \rightarrow B$ is a D-formula;

iv) If A is a D-formula then $\neg A$ is a D-formula.

The Theorem Prover Dialogue Rules, from now on Dialogue Rules, are a subset of the rules previously shown in § 4 . Actually they are Rule Identity, Rule Ground in three different versions, a variant of Rule Withdrawal specifically tailored to support backtracking and finally the Rule 'Reductio ad absurdum'.

Rule Identity works exactly as it was previously described.This rule will be used in this context to start the dialogue/proof through the introduction of the theorem that we want to prove. In case of success (the original theorem was proved), the rule will be used again in order to 'confirm' the theorem.

The versions of Rule Ground can be divided into two subsets. The first one consists only of Rule AC (Answer-to-Challenge) that is a version of Rule Ground for atomic challenges. The second subset is called Connective Rules. It is composed of two rules that are like Rule AC with the additional power of dealing with the connectives - and & implication - that appear inside the challenge.

The theorem prover needs a rule that will support a kind of backtracking. Mackenzie's Withdrawal is slightly changed in order to eliminate both actual answer of the challenge and the conditional generated by it. Rule WGY (Withdrawal of the Answer of a Challenge) was tailored specifically for this task.

The Rule 'Reductio ad absurdum' needs further explanation. Our classical Theorem Prover first will try a constructive (intuitionistic) proof. Failing in this attempt, this rule will be triggered in order to restart the proof in a 'classical' way.

TP1) Rule Identity

$$(B{:}b)\ (A{:}a) \Rightarrow [\text{Agent}, \mathbf{I}(s)]\ (B{:}b + \{s\})\ (A{:}a + \{s\})$$

After Agent has asserted {s}; {s} is added to both commitment stores.

TP2) Rule AC (Answer-to-Challenge)

$$(B{:}b)(A{:}a) \Rightarrow [B, \mathbf{Y}(s)][A, \mathbf{G}(t)]\ (B{:}b - \{s\} + \{\mathbf{Y}(s), t{\rightarrow}s, t\})(A{:}a + \{s, t{\rightarrow}s, t\})$$

After B has challenged {s}, {s} is removed from (B:) and {**Y**(s)} is added to it and {s} is added to (A:). Given that {t} is an answer to the challenge, A answers {t} and then {t→s, t} are added to both commitment stores.

TP3) Rule ACI (Answer to the Challenge of an Implication)

$$(B{:}b)\ (A{:}a) \Rightarrow [B,\ \mathbf{Y}(p{\rightarrow}q)][A,\ \mathbf{G}([p],\ q)]$$
$$(B{:}b - \{p{\rightarrow}q\} + \{\mathbf{Y}(p{\rightarrow}q)\} + \{[p],\ q\})(A{:}a + \{p{\rightarrow}q,\ [p],\ q\})$$

After B has challenged {p→q}, {p→q} is removed from (B:) and {**Y**(p→q)} is added to it and {p→q} is added to (A:). A answers {[p], q} and then {[p], q} are added to both commitment stores. The meaning of '[p]' is that 'p' becomes an assumption and can be used later inside the proof. This rule captures the inference rule '→Introduction' [A];B/A→B.

TP4) Rule ACA (Answer to the Challenge of an And)

$$(B{:}b)\ (A{:}a) \Rightarrow [B,\ \mathbf{Y}(p{\wedge}q)][A,\ \mathbf{G}(p,\ q)]$$
$$(B{:}b - \{p{\wedge}q\} + \{\mathbf{Y}(p{\wedge}q)\} + \{p,\ q\})\ (A{:}a + \{p{\wedge}q,\ p,\ q\})$$

After B has challenged {p∧q}, {p∧q} is removed from (B:) and {**Y**(p∧q)} is added to it and {p∧q} is added to (A:). A answers {p, q} and then {p, q} are added to both commitment stores. This rule captures the inference rule '∧ Introduction' A,B/∧ B.

TP5) Rule WGY (Withdrawal of the Answer of a Challenge)

$$(B{:}b,\mathbf{Y}(s),\ t{\rightarrow}s,\ t)\ (A{:}a,\ s,\ t{\rightarrow}s,\ t)$$
$$\Rightarrow [B,\ \mathbf{Y}(t)]\ [A,\ \mathbf{W}(\mathbf{G}(\mathbf{Y}(s)))]\ (B{:}b,\mathbf{Y}(s),\ t{\rightarrow}s + \{\mathbf{Y}(t)\})\ (A{:}a)$$

After B has challenged {t},{t} is removed from (B:b,**Y**(s), t→s, t) and {**Y**(t)} is added to it and {t} is added to (A:a,s,t→s, t). Given that there is no answer to the challenge {**Y**(t)},A answers {**W**(**G**(**Y**(s)))} and then {t→s, t} is removed from (A:a,s, t→s, t) resulting in (A:a,s).

TP6) Rule 'Reductio ad absurdum'

$$(B{:}\mathbf{Y}(t),b)\ (A{:}t,a) \Rightarrow\ [B,\ \mathbf{Y}(s)]\ [A,\ \mathbf{G}(\neg t{\wedge}t)]$$
$$(B{:}\mathbf{Y}(t),b + \{(\neg t{\wedge}t){\rightarrow}s)\ (A{:}t,\ [\neg t],a + \{(\neg t{\wedge}t){\rightarrow}s)\}$$

After B has challenged {s}, {s} is removed from (B:**Y**(t),b) and {**Y**(s)} is added to it and {s} is added to (A:t,a). Given that there is no answer to the challenge {**Y**(s)} and that all attempts of backtracking have failed, A answers {(¬t∧t)} and then {[¬t]} is inserted into (A:) just after the original theorem {t}resulting in (A:t,[¬t],a) and then {(¬t∧t)→s} is added to both commitment stores.

So far we have described the Theorem Prover, we have shown what is a proof in this framework, but nothing has been done to actually obtain the dialogue that proves a given theorem. In this section we give the directives to automatically produce a dialogue-proof of a formula.

The connective rules TP2 and TP3 are deterministics. Whenever an implicational or a connective formula is challenged, they automatically provide an answer for the challenge. On the other hand, if the challenged formula is either an atom or a negated one, the Theorem Prover uses the rule AC together with some heuristics that are shown below.

Challenging an atom:

1) If there is one formula in the Database of the form $B \rightarrow A$, answer B.

2) If there is one formula in the Database of the form $\neg Q$, answer $Q \wedge \neg Q$.

3) If there is one formula in the Database of the form $B \rightarrow \neg Q$,answer $Q \wedge \neg Q$.

4) RESTART the proof; Empty both commitment stores, challenge the original theorem Y again. Answer $\neg Y \rightarrow Y$.

obs.: The case 4) is only valid for classical proofs. This is essential to differentiate between the classical and the intuitionistic Theorem Prover.

Challenge of a negated formula $\neg Q$:

1) Answer $Q \rightarrow \neg Q$.

2) If Q is an atom and there is one formula in the Database of the form $B \rightarrow \neg Q$, answer B.

3) If there is one formula in the Database of the form $\neg Q$, answer $Q \wedge \neg Q$.

4) If there is one formula in the Database of the form $B \rightarrow \neg Q$, answer $Q \wedge \neg Q$.

7 Using the Theorem Prover

From now on Agent-A will be called **V** for verifier and its commitment store **V:** for verification. Agent-B will be called **S** for heuristic searcher and its commitment store **S:** for search. Commitment store (**S:**) is the bookkeeping of all attempts to prove the original theorem and (**V:**) will have the 'proof'. The 'proof' will appear in a goal directed fashion.

1 Given DATABASE prove GOAL	
DATABASE	GOAL
i) b^d i i) a^b i i i) a	d

	'S'	'V'	'(S:)'	'(V:)'
0)			«	«
1)		I(d)	d	d
2)	Y(d)		Y(d)	
3)		G(b)	b^d	b^d
			b	b
4)	Y(b)		Y(b)	
5)		G(a)	a^b	a^b
			a	a
6)	I(d)		d	

0) Both commitment stores begin null.

1) **V** triggers the proof asserting the original theorem **I(d)**.

2) **S** challenges the original theorem **Y(d)**.

3) **V** looks for fact **d** in the database or as an assumption. Failing it starts the search for the first clause in the database which has **d** as head $\rightarrow$**d**. It finds **b**$\rightarrow$**d** thus answering **G(b)**.

4) **S** cannot challenge **b**$\rightarrow$**d** because it is in database therefore it challenges **Y(b)**.

5) **V** looks for fact **b** in the database or as an assumption. Failing it starts the search for the first clause in the database which has **b** as head .$\rightarrow$**b**. It finds **a**$\rightarrow$**b** thus answering **G(a)**.

6) Given that **S** neither can challenge **a**$\rightarrow$**b** nor **a** because they are both in the database, it asserts the original theorem **I(d)** concluding the proof.

2 Given DATABASE prove GOAL	
DATABASE i) a	GOAL▪ □□a

	'S'	'V'	'(S:)'	'(V:)'
0)			«	«
1)		I(□□a)	□□a	□□a
2)	Y(□□a)		Y(□□a)	
3)		G(□a^□□a)	(□a^□□a)^□□a	(□a^□□a)^□□a
			□a^□□a	□a^□□a
4)	Y(□a^□□a)		Y(□a^□□a)	
5)		G([□a],□□a)	[□a],□□a	[□a],□□a
6)	Y(□□a)		Y(□□a)	
7)		G(a∞□a)	(a∞□a)^□□a	(a∞□a)^□□a
			a∞□a	a∞□a
8)	Y(a∞□a)		a∞□a	a∞□a
9)		G(a,□a)	a,□a	a,□a

3 Given DATABASE prove GOAL	
DATABASE i) a^b ii) c^b iii)c	GOAL b

'S'	'V'	'(S:)'	'(V:)'
0)		«	«
1)	I(b)	b	b
2)Y(b)		Y(b)	
3)	G(a)	a^b	a^b
		a	a
4)Y(a)		Y(a)	
5)	W(G(Y(b)))		
6)Y(b)		Y(b)	
7)	G(c)	c^b	c^b
		c	c
8)I(b)		b	

4 Given DATABASE prove GOAL			
DATABASE			GOAL ▪
i) (p^q)^p			p
'S'	'V'	'(S:)'	'(V:)'
0)		«	«
1)	I(p)	p	p
2) Y(p)		Y(p)	
3)	G(p^q)	(p^q)^p	(p^q)^p
		p^q	p^q
4) Y(p^q)		Y(p^q)	
5)	G([p],q)	[p],q	[p],q
6) Y(q)		Y(q)	
7) G(¬p∞p)			
	Rule "Reductio ad absurdum"		
			[¬p]
		Y(p)	
		(p^q)^p	(p^q)^p
		p^q	p^q
		Y(p^q)	
		[p],q	[p],q
		Y(q)	q
		(¬p∞p)^q	(¬p∞p)^q
		¬p∞p	¬p∞p
8) Y(¬p∞p)		¬p∞p	¬p∞p
9)	G(¬p,p)	¬p,p	¬p,p
10)I(p)		p	

8 Conclusion and Further Work

Dialogue seems to provide a satisfactory formal basis for such a framework. We are currently creating an environment based on dialogue in order to implement this theorem prover. Natural extensions to this work could be among others the lifting of the theorem prover for first-order and the inclusion of modalities. We are also using this paradigm for software development methods and software specification.

Acknowledgements

The authors would like to thank Tom Maibaum, Dov Gabbay, Anthony Finkelstein, Marcos Costa, Celso Niskier and João Gondim for their contribution to this work. Hugo Fuks is supported by the Brazilian national research council CNPq, grant 202471/86-CC. Marcelino C. Pequeno is supported by the 'Universidade do Estado do Ceará' and by the Brazilian Ministry of Education research council CAPES, grant 6231/84-4.

References

[Fin87] Finkelstein, A., Fuks, H.,Niskier, C., Sadler, M. (1987); A dialogic framework for software development; Research Report DoC 87/15; Department of Computing, Imperial College of Science and Technology, London.

[Gab84] Gabbay, D. M. (1984); Elementary Logic: A Procedural Perspective; Lecture Notes, Logic 141, Draft 1984, Department of Computing, Imperial College of Science and Technology, London.

[Gol82] Goldblatt, R. (1982); Axiomatising the Logic of Computer Programming. XI; Lectures Notes in Computer Science Vol . 130, Springer-Verlag.

[Ham71] Hamblin, C. (1971); Mathematical models of dialogue; Theoria, V2, pp130-155.

[Hin73] Hintikka, J. (1973); Logic, Language-Games, and Information, Clarendon Press, Oxford.

[Mac79a] Mackenzie, J. (1979a); How to stop talking to tortoises; Notre Dame Journal of Formal Logic, V20, pp 705-717.

[Mac79b] Mackenzie, J. (1979b); Question begging in non-cumulative systems; Journal of Philosophical logic, V8, pp 117-133.

[Mac80] Mackenzie, J. (1980); Why do we number theorems?; Australasian Journal of Philosophy, V58, pp 135-149.

[Mac81] Mackenzie, J. (1981); The Dialectics of Logic; Logique et Analyse, V24, pp 159-177.

[Mac84] Mackenzie, J. (1984); Begging the question in dialogue; Australasian Journal of Philosophy, V61, pp 174-181.

[Mac85] Mackenzie, J. (1985); No Logic before Friday; Synthese, V63, pp 329-341.