

System description: CondLean

Nicola Olivetti, Gian Luca Pozzato

Dip. di Informatica, Università degli studi di Torino

Abstract. In this paper we present a theorem prover called CondLean for normal propositional conditional logics CK, CK+ID, CK+MP and CK+MP+ID. The theorem prover implements some sequent calculi for these logics recently introduced. The theorem prover is developed following the methodology of Lean-TAP and it is implemented in SICStus Prolog. The theorem prover also comprises a graphical user interface implemented in JAVA language. CondLean can be downloaded at the site www.di.unito.it/~olivetti/CONDLEAN/

1 Introduction

Conditional logics have a long history. Recently, they have been used in computer science and artificial intelligence, for example in knowledge representation, non-monotonic reasoning, deductive databases, belief revision and natural language semantics. In spite of their significance, very few proof systems have been proposed for these logics. We use selection function semantics, where the selection function f selects, for a world w and a formula A , the set of worlds $f(w, A)$ which are "most similar to w " or "closer to w " given the information A . A conditional sentence $A \Rightarrow B$ is true in a world w whenever B is true in every world selected by f for w and A .

In [1] sequent calculi SeqS¹ are introduced for minimal conditional logic **CK** and for three extensions of it, namely **CK+ID**, **CK+MP** and **CK+MP+ID**. The calculi make use of labels to represent possible worlds and transition formulas to represent selection function's values: for example, transition $x \xrightarrow{A} y$ is used to represent that $y \in f(x, A)$.

In this work we describe an implementation in SICStus Prolog of SeqS calculi for $\text{CK}\{+R\}$ ². This program gives a decision procedure for these logics; as far as we know this is the first theorem prover for these logics.

For each system, we introduce three different versions:

1. a simple version, where Prolog *constants* are used to represent SeqS's labels;
2. a more efficient one, where labels are represented by Prolog *variables*, inspired by the free-variable tableau introduced in [2];
3. a "two-phase" theorem prover, which first attempts to prove a sequent by using an incomplete proof procedure (phase 1); the theorem prover uses the free-variable version in phase 2 if in phase 1 has reported a failure.

¹ S stands for CK, ID, MP or ID+MP.

² R stands for ID, MP or MP+ID.

Users can interact with the theorem prover through a **graphical interface**, implemented in *Java*, using `se.sics.jasper` package to link this component to the SICStus Prolog kernel.

2 Conditional logics **CK**, **CK+ID**, **CK+MP**, **CK+MP+ID** and the sequent calculi **SeqS**

Conditional logics are extensions of logic by the conditional operator \Rightarrow . We consider a propositional language \mathcal{L} over a set of propositional variables ATM . Formulas of \mathcal{L} are built from propositional variables by means of the boolean operators \rightarrow, \perp and the conditional operator \Rightarrow . As explained above, we adopt here the so-called propositional selection function semantics [3]. A selection function model for \mathcal{L} is a triple $\mathcal{M} = \langle \mathcal{W}, f, [\] \rangle$, where \mathcal{W} is a non-empty set of items called *worlds*, f is a function of type $f : \mathcal{W} \times 2^{\mathcal{W}} \rightarrow 2^{\mathcal{W}}$, called the *selection* function; $[\]$ is an evaluation function of type $ATM \rightarrow 2^{\mathcal{W}}$. $[\]$ assigns to an atom p the set of worlds where p is true. The evaluation function $[\]$ can be extended to every formula by means of the following inductive clauses:

1. $[\perp] = \emptyset$;
2. $[A \rightarrow B] = (\mathcal{W} - [A]) \cup [B]$;
3. $[A \Rightarrow B] = \{w \in \mathcal{W} \mid f(w, [A]) \subseteq [B]\}$.

We say that a formula A is *valid* in a model \mathcal{M} as above if $[A] = \mathcal{W}$. A formula A is *valid* (denoted by $\models_{CK} A$) if it is valid in every model \mathcal{M} .

The above is the semantics of the basic conditional logic **CK**, where no specific properties are assumed on the selection function f . Moreover, we consider also the following extensions of **CK**, namely **CK+ID**, **CK+MP**, and **CK+ID+MP**. The semantic conditions (MP) and (ID) corresponds to well-known axioms shown in the following table.

<i>System</i>	Axiom	Model condition
(ID)	$A \Rightarrow A$	$f(x, [A]) \subseteq [A]$
(MP)	$(A \Rightarrow B) \rightarrow (A \rightarrow B)$	$w \in [A] \rightarrow w \in f(w, [A])$

In **Figure 1** we present the calculi for $CK\{+R\}$ introduced in [1]:

$\text{(AX)} \Gamma, F \vdash \Delta, F$	$\text{(A}\perp\text{)} \Gamma, x : \perp \vdash \Delta$
$\text{(ContrL)} \frac{\Gamma, F, F \vdash \Delta}{\Gamma, F \vdash \Delta}$	$\text{(ContrR)} \frac{\Gamma \vdash \Delta, F, F}{\Gamma \vdash \Delta, F}$
$\text{(\(\rightarrow\ R)} \frac{\Gamma, x : A \vdash x : B, \Delta}{\Gamma \vdash x : A \rightarrow B, \Delta}$	$\text{(\(\rightarrow\ L)} \frac{\Gamma \vdash x : A, \Delta \quad \Gamma, x : B \vdash \Delta}{\Gamma, x : A \rightarrow B \vdash \Delta}$
$\text{(EQ)} \frac{u : A \vdash u : B \quad u : B \vdash u : A}{\Gamma, x \xrightarrow{A} y \vdash x \xrightarrow{B} y, \Delta}$	
$\text{(\(\Rightarrow\ L)} \frac{\Gamma \vdash x \xrightarrow{A} y, \Delta \quad \Gamma, y : B \vdash \Delta}{\Gamma, x : A \Rightarrow B \vdash \Delta}$	$\text{(\(\Rightarrow\ R)} \frac{\Gamma, x \xrightarrow{A} y \vdash y : B, \Delta}{\Gamma \vdash x : A \Rightarrow B, \Delta} \quad (y \notin \Gamma, \Delta)$
$\text{(ID)} \frac{\Gamma, y : A \vdash \Delta}{\Gamma, x \xrightarrow{A} y, \vdash \Delta}$	$\text{(MP)} \frac{\Gamma \vdash x : A, \Delta}{\Gamma \vdash x \xrightarrow{A} x, \Delta}$

Fig. 1. Sequent calculi SeqS; the (ID) rule is for SeqID and SeqID+MP only; the (MP) rule is for SeqMP and SeqID+MP only.

The calculi make use of labelled formulas, where labels are drawn from a denumerable set \mathcal{A} ; there are two kinds of formulas:

1. *labelled formulas*, denoted with $x : A$, where $x \in \mathcal{A}$ and A is a propositional conditional formula;
2. *transition formulas*, denoted with $x \xrightarrow{A} y$, where $x, y \in \mathcal{A}$ and A is a propositional conditional formula; we use a transition formula $x \xrightarrow{A} y$ to represent that $y \in f(x, [A])$.

We define sequent validity as follows:

Definition: sequent validity: given a model $\mathcal{M} = \langle \mathcal{W}, f, [\] \rangle$ for \mathcal{L} , and a label alphabet \mathcal{A} , we consider any *mapping* $I : \mathcal{A} \rightarrow \mathcal{W}$. Let F be a labelled formula, we define $\mathcal{M} \models_I F$ as follows

$$\mathcal{M} \models_I x : A \text{ iff } I(x) \in [A] \quad \text{and} \quad \mathcal{M} \models_I x \xrightarrow{A} y \text{ iff } I(y) \in f(I(x), [A])$$

We say that $\Gamma \vdash \Delta$ is *valid* in \mathcal{M} if for every mapping $I : \mathcal{A} \rightarrow \mathcal{W}$, if $\mathcal{M} \models_I F$ for every $F \in \Gamma$, then $\mathcal{M} \models_I G$ for some $G \in \Delta$. We say that $\Gamma \vdash \Delta$ is valid in that system³ if it is valid in every \mathcal{M} .

³ As we explained above, restrictions on the selection functions f determine the conditional system.

Theorem (soundness and completeness) [1]: a sequent $\Gamma \vdash \Delta$ is valid if and only if $\Gamma \vdash \Delta$ is derivable in SeqS.

As usual, in order to obtain a decision procedure we have to control the application of the contraction rules in a backward proof search of a sequent derivation. To this regard we have the following results:

Theorem 2 [1], [4] : if $\Gamma \vdash \Delta$ is derivable using SeqCK (resp. SeqID), it has a derivation where there are no applications of (Contr L) and (Contr R).

In contrast, *we cannot eliminate contractions in SeqMP and SeqID+MP*; more precisely, in these calculi we do not need (Contr R), but we might need use (Contr L) on conditional formulas $x: A \Rightarrow B$ to preserve the completeness of the calculi. For example, $x: \top \Rightarrow ((B \rightarrow (\top \Rightarrow B)) \rightarrow \perp) \vdash$ is valid in CK+MP, but we need to apply (Contr L) with constituent $x: \top \Rightarrow ((B \rightarrow (\top \Rightarrow B)) \rightarrow \perp)$ to have a derivation of it. However, we can show a limitation on the application of left contraction on conditional formulas, as follows:

Theorem 3 [4] : if $\Gamma \vdash \Delta$ is derivable in SeqMP (resp. SeqID+MP), then it has a derivation where there are no applications of (Contr R) and there is at most one application of (Contr L) with constituent $x: A \Rightarrow B$ in *each branch* of the proof tree, for each conditional formula $x: A \Rightarrow B$.

These results give a constructive proof of decidability of the respective systems, alternative to the semantic proof of decidability based on the finite model property.

One useful property of the calculi is that all rules except $(\Rightarrow L)$ and (EQ) (obviously) *permute over all the others*; as a consequence we have that backtracking points will be introduced only by these rules. This property is important for the implementation of the sequent calculi.

3 An implementation of SeqS

In this section we show an implementation of the sequent calculi SeqS; it is a SICStus Prolog program inspired by leanTAP, introduced in [5]; the source code is a set of clauses just implementing the sequent calculi, exploiting Prolog's built-in depth first search mechanism.

We represent every component of a sequent (antecedent and consequent) with a **list** of labelled formulas, partitioned into three sub-lists: atomic formulas (literals), transition formulas and complex formulas (not atomic). Atomic and complex formulas are represented by a list like $[x, a]$, where x is a Prolog constant and a is the formula. A transition formula $x \xrightarrow{A} y$ is encoded as $[x, a, y]$.

As we explained above, we show three different implementations. The first one,

called **constant labels**, uses Prolog constants to represent SeqS's labels. The sequent calculi is implemented by the predicate

```
prova(Sigma, Delta, Labels)
```

This predicate succeeds if and only if $\Sigma \vdash \Delta$ is derivable in SeqS. **Sigma** and **Delta** are the lists representing multisets Σ and Δ , respectively; **Labels** is the list of labels introduced in that branch. For example, to prove $x: A \Rightarrow (B \wedge C)$ ⁴ $\vdash x: A \Rightarrow B, x: C$ in CK, one queries CondLean with the following goal:

```
prova([], [], [[x, a => (b and c)]]], [[x,c], [], [[x, a =>
b]]], [x])
```

All the clauses of **prova** implement SeqS's axioms or rules, except for contractions⁵; for examples, we show clause implementing (A \perp) and clause implementing (\Rightarrow L):

```
prova([LitSigma, -, -], -, -):-
member([_,false], LitSigma), !.

prova([LitSigma,TransSigma,LabSigma],[LitDelta,TransDelta,LabDelta],
Labels):-
select([X,A=>B],LabSigma,ResLabSigma),
member(Y,Labels),
colloca6([Y,B],LitSigma,ResLitSigma,NewLitSigma,NewLabSigma),
prova([LitSigma,TransSigma,ResLabSigma],[LitDelta,
[X,A,Y]|TransDelta],LabDelta,Labels),
prova([NewLitSigma,TransSigma,NewLabSigma],[LitDelta,
TransDelta,LabDelta],Labels).
```

It is easy to understand how this implementation tries to find a derivation for a sequent $\Sigma \vdash \Delta$: first of all, if $\Sigma \vdash \Delta$ is an axiom, the clauses implementing axioms make the goal succeeds. If it is not, theorem prover's computation goes on using the first rule applicable to that sequent: if **LabSigma** contains a formula $[X, A \text{ and } B]$, then the clause implementing (\wedge L) rule is applied, invoking **prova** on the only premise of (\wedge L); else, if **LabDelta** contains a formula $[X, A \text{ or } B]$, then the clause implementing (\vee R) rule is applied, invoking **prova** on its premise, and so on for the other rules, ordered to posticipate the application of branching rules.

When a (\Rightarrow L) clause is applied to prove $\Sigma \vdash \Delta$, a backtracking point is introduced by choosing a label **Y** to call **prova** on the premises; when we can use more

⁴ CondLean extends the sequent calculi to formulas containing \neg, \wedge, \vee and \top .

⁵ In SeqMP and SeqID+MP (ContrL) is "embedded" in (\Rightarrow L), although in a controlled way in light of Theorem 3 above.

⁶ Used to introduce the sub-formula $[Y, B]$ respecting the partition of the sequent in atomic, transition and complex formulas.

than one label to apply (\Rightarrow L), Prolog's built-in backtracking leads to try all possible derivations, in case of failure instantiating the rule with any available label.

Choosing, sooner or later, the right label to apply (\Rightarrow L) could strongly influence CondLean's performances: suppose that we can use n labels to prove (\Rightarrow L)'s premises on Σ' , $x: A \Rightarrow B \vdash \Delta$, but only one of them lets the theorem prover to find a derivation of the sequent and this label is selected after *all the other $n-1$ labels*. System can say 'yes' only after $n-1$ backtrackings, degrading performances.

The second version of our implementations, called **free-variable**, uses a *Prolog variable to represent all the labels that the theorem prover could use to apply (\Rightarrow L)*: proving $\Sigma', 1: A \Rightarrow B \vdash \Delta$ ⁷, the theorem prover will call `prova` on the following premises: $\Sigma' \vdash \Delta$, $1 \xrightarrow{A} V$ and $V: B, \Sigma' \vdash \Delta$, where V is a Prolog variable which will be instantiated by Prolog's pattern matching to apply (EQ) rule or to *close a branch with an axiom*. More precisely, computation goes on without instantiating free-variable V , which is used as a parameter; when the theorem prover will try to compute `prova` on a sequent like $V: F, \Psi \vdash n: F, \Psi'$, clauses implementing axioms will end the proof search on that branch: in fact, $V: F, \Psi \vdash n: F, \Psi'$ is an axiom with the substitution of the free-variable V with the constant label n .

We show the clause implementing (\Rightarrow L): library `clpfd`'s predicates are used to manage free-variable domains:

```
prova([LitSigma,TransSigma,LabSigma],[LitDelta,
    TransDelta,LabDelta],Max):-
    select([X,A => B],LabSigma,ResLabSigma),
    domain([Y],1,Max),
    Y#>X8,
    colloca([Y,B],LitSigma,ResLabSigma,NewLitSigma,NewLabSigma),
    prova([NewLitSigma,TransSigma,NewLabSigma],
        [LitDelta,TransDelta,LabDelta],Max),
    prova([LitSigma,TransSigma,ResLabSigma],[LitDelta,
        [X,A,Y]|TransDelta,LabDelta],Max).
```

On a sequent with 65 labels on the antecedent this version finds a solution in 460 mseconds, whereas the constant labels version takes 4326 mseconds.

We developed a third version, called **euristic version**, making a "two-phase" computation: during "Phase 1" an *incomplete* theorem prover tries to find a derivation exploring a *reduced search space*, which takes a very small time to compute; in case of failure, the free-variable version is used to search a deriva-

⁷ This version represents labels with integers, starting with 1.

⁸ In SeqCK and SeqID we can only use labels introduced *after* label `X`, thus we introduce this constraint. In SeqMP and SeqID+MP we use the following condition: `Y#>=X`.

tion for that sequent.

For SeqMP and SeqID+MP implementation, as we mentioned in previous section, the theorem prover must use (Contr L) applied *at most once for each formula* $x: A \Rightarrow B$ on each branch of the derivation. Our solution consists in contracting every instance of $x: A \Rightarrow B$ one time in every branch, introducing another argument to the `prova` predicate:

```
prova(Sigma, Delta, Labels, CondContr)
```

When $(\Rightarrow L)$ is applied to a formula $x: A \Rightarrow B$, also the formula is duplicated (contracted) into `CondContr` list; $(\Rightarrow L)$ is also applied to formulas in `CondContr`: in this case, the principal formula is *not* duplicated.

4 The application CondLean

Our theorem prover **CondLean** has a graphical interface, written in Java, which allows users to ask a derivation of a sequent in CK or in one of its extensions, choosing also the version of the theorem prover (constant labels, free-variable, heuristic version). The graphical interface interacts with the SICStus Prolog implementation using package `se.sics.jasper`'s classes: user can submit a sequent only editing it in a text field and clicking a button; then, the control window will show the result of the proof search. When a sequent is valid in the conditional system chosen by the user, the application lets user to click a button to view the proof tree of the sequent in a special window, or to build a latex file containing the proof tree or to view a list of statistics about the proof tree.

Users do not need to know how to call the predicate `prova`: this is hidden by the graphical user interface, which interacts with the reasoner.

In **Appendix** we show some pictures from CondLean.

5 Statistics, conclusions and future work

In section 3 we introduced the three versions of SeqS; all these implementations seems to run fast, even on a very small machine⁹. More precisely, we tested SeqCK implementation with the following results: constant label version succeeds in 79 tests on 90 in less than two seconds; free variables succeeds on 73 sequents in the same time (but 67 in less than 10 mseconds); heuristic version is able to show derivability of 78 CK theorems in less than 2 seconds (70 in less than 500 mseconds). The test samples are obtained modifying samples from [2].

Studying performances relatively sequent degree, defined as the maximum nesting level of \Rightarrow operator, we found the following statistics:

⁹ These results are obtained running SICStus Prolog 3.10.0 on a Pentium 166 MMX, 96 MB RAM machine.

Degree	2	6	9	11	15
Time(msec)	5	500	650	1000	2000

In future research we intend to expand our work in three directions: 1. extend the complexity analysis to other conditional systems; 2. find labelled analytic calculi for other conditional logics; 3. upgrade the graphical user interface to have better performances in displaying a proof tree (actually, it requires some minutes to display a proof tree with more than 70 nodes).

References

1. N. OLIVETTI, C.B. SCHWIND. *A calculus and complexity bound for minimal conditional logic*. PROC. ICTCS01, LNCS 2202, 2001.
2. B. BECKERT, R. GORÈ. *Free Variable Tableaux for Propositional Modal Logics*. TABLEAUX-97, LNCS 1227, SPRINGER, 1997.
3. D. NUTE. *Topics in conditional logic*. REIDEL, 1980.
4. G. L. POZZATO. *Deduzione automatica per logiche condizionali: analisi e sviluppo di un theorem prover*. TESI DI LAUREA, INFORMATICA, UNIVERSITÀ DEGLI STUDI DI TORINO, IN PREPARAZIONE, 2003.
5. B. BECKERT, J. POSEGGA. *leanTAP: Lean tableau-based deduction*. JOURNAL OF LOGIC PROGRAMMING, 28, 1996.

Appendix: some pictures from CondLean

