Issues in Machine-checking the Decidability of Implicational Ticket Entailment

Jeremy E. Dawson* and Rajeev Goré

Research School of Computer Science, Australian National University

Abstract. The decidability of the implicational fragment $T_{\rightarrow}$ of the relevance logic of ticket entailment was recently claimed independently by Bimbó and Dunn, and Padovani. We present a mechanised formalisation, in Isabelle/HOL, of the various proof-theoretical results due to Bimbó and Dunn that underpin their claim. We also discuss the issues that stymied our attempt to verify their proof of decidability.

1 Introduction

Sequent calculi are useful in many areas of logic, particularly for decidability arguments. Here, we consider the complications that arise when dealing with a substructural logic where one or more of the rules of associativity, commutativity, weakening and contraction are missing. We focus on the implicational fragment $T_{\rightarrow}$ of the substructural logic of "ticket entailment", recently claimed as decidable independently by Bimbo and Dunn [BD13], and by Padovani [Pad11].

As is well-known, pen-and-paper proofs about sequent calculi are notoriously tedious and error-prone [GR12], particularly when the authors elide proofs because "the proof is similar". The proofs of Bimbo and Dunn are intricate, some requiring a triple induction over the "grade", "height" and "contraction degree" of the instance of cut. They state in a footnote that these complicated inductions appear to be necessary [BD12, footnote 9]. Moreover, they use the "dangerous" phrases described above, so how can we be sure that their proofs are sound?

To check, we first formulate the various sequent and "consecution" calculi from [BD12,BD13]. We then describe how we encoded these calculi into the interactive proof-assistant Isabelle/HOL and how we mechanised the various proof-theoretical results of these various calculi. We then explain the issues that stymied our attempt to verify the proof of their main theorem in Isabelle/HOL.

Previously, we have machine-checked various types of calculi: multiset-based sequent calculi with explicit structural rules [DG10], display calculi [DG02], and (shallow and deep) nested sequent calculi [DCGT14]. Here, we needed two novelties: singletons on the right and (non-display) "consecution" calculi built from "structures" (binary trees) where all internal nodes contain a non-commutative and non-associative binary operator " ; " and where all leaves are formulae.

Notation: we use A, B, C for formulae, use Γ, Γ_1, Γ_2 for multisets, use U, V, X, Y, Z for structures, and use $X\{Y\}$ instead of $\mathfrak{A}[\mathfrak{B}]$. We use π and τ for the transformations on derivations, but use δ for derivations instead of Δ.

* Supported by Australian Research Council Discovery Grant DP120101244.

Name	Axioms	Logic			
		$T_{\to}$	$T^{\mathbf{t}}_{\to}$	$R_{\to}$	$R^{\mathbf{t}}_{\to}$
(A1)	$A \to A$	✓	✓	✓	✓
(A2)	$(A \to B) \to (C \to A) \to (C \to B)$	✓	✓	✓	✓
(A3)	$(A \to B \to C) \to (B \to A \to C)$			✓	✓
(A4)	$(A \to A \to B) \to (A \to B)$	✓	✓	✓	✓
(A5)	$(A \to B) \to (B \to C) \to (A \to C)$	✓	✓		
Name	Rules of Inference				
(R1)	from $A \to B$ and A, deduce B	✓	✓	✓	✓
(R2)	$\vdash A$ // $\vdash \mathbf{t} \to A$		✓		✓

Fig. 1. Axiomatisations of various logics

2 Summary of Various Calculi of Bimbó and Dunn

The formulae of our logics are built from an infinite supply of atomic formulae using the BNF grammar below where p is any atomic formula and $\mathbf{t}$ is a constant:

$$A ::= \quad p \mid \mathbf{t} \mid A \to A$$

The superscript $\mathbf{t}$ determines whether or not the verum constant $\mathbf{t}$ is in the syntax. As usual, we drop parentheses and write $A \to B \to C$ for $A \to (B \to C)$. The various logics are defined in Figure 1 [BD12].

A sequent $\Gamma \vdash C$ consists of a finite, possibly empty, multiset Γ of formulae and a formula C. We prefer Greek letters in keeping with modern usage in sequent calculi. The specific sequent calculi that we deal with are tabled in Figure 2. The "consecution" calculi of Bimbó and Dunn use structures where: every formula is a structure, and if X and Y are structures then so is $(X\ ;\ Y)$. Note: there is no empty structure [BD12]! A consecution $X \vdash C$ consists of a structure X and a formula C. We write $X\ ;\ Y\ ;\ Z$ for $((X\ ;\ Y)\ ;\ Z)$ [BD12].

A structure is thus a binary tree where all internal nodes contain " ; " and the leaves contain formulae. Suppose X is such a structure (tree) and let Y be the substructure that appears at some particular node in this tree: written $X\{Y\}$. If we now replace this occurrence of Y by an occurrence of the structure Z, we obtain the structure $X\{Z\}$. In the rules shown in Figure 3, the premises locate the node at which a particular substructure appears in a larger structure. The conclusion shows the result of replacing the substructure occurrence at that node by some structure occurrence, as just described. We use $X\{Y\}$ instead of the $X[Y]$ used by Bimbó and Dunn since the latter can cause confusion with the use of brackets to capture limited contraction in the $[{\to}{\vdash}]$-rule from Figure 2.

3 Our Isabelle mechanisation

Our mechanisation builds on our previous work on mechanising traditional sequent calculi [DG10]. That work is a deep embedding of rules and of the variables in them, which permits explicit substitution functions for the variables in a small

$$\text{(id)}\ \frac{}{A \vdash A} \qquad (\to\vdash)\ \frac{\Gamma_1 \vdash A \qquad B, \Gamma_2 \vdash C}{\Gamma_1, A \to B, \Gamma_2 \vdash C} \qquad (\vdash\to)\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

$$(\text{W}\vdash)\ \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C} \qquad (\mathbf{t}\vdash)\ \frac{\Gamma \vdash C}{\mathbf{t}, \Gamma \vdash C} \qquad (\vdash \mathbf{t})\ \frac{}{\vdash \mathbf{t}}$$

$$[\to\vdash]\ \frac{\Gamma_1 \vdash A \qquad B, \Gamma_2 \vdash C}{[\Gamma_1, A \to B, \Gamma_2] \vdash C}\ \dagger$$

† is the condition that $[\Gamma_1, A \to B, \Gamma_2]$ is a sub-multiset of $\Gamma_1, A \to B, \Gamma_2$ such that:

(a) $A \to B$ occurs at least once in $[\Gamma_1, A \to B, \Gamma_2]$ but may have 0, 1 or 2 fewer occurrences than in $\Gamma_1, A \to B, \Gamma_2$

(b) if D, which is distinct from $A \to B$, occurs in Γ_1, Γ_2 then D occurs at least once in $[\Gamma_1, A \to B, \Gamma_2]$ but may have 0 or 1 fewer occurrences than in Γ_1, Γ_2.

	(id)	$(\to\vdash)$	$(\vdash\to)$	(W⊢)	$(\mathbf{t}\vdash)$	$(\vdash \mathbf{t})$	$[\to\vdash]$
$LR_\to$	✓	✓	✓	✓			
$LR^{\mathbf{t}}_\to$	✓	✓	✓	✓	✓	✓	
$[LR_\to]$	✓	✓	✓				✓
$[LR^{\mathbf{t}}_\to]$	✓	✓	✓		✓	✓	✓

Fig. 2. Various Sequent Rules

finite set of rules: see [DG10] for our understanding of what this means, and for more details. Here, we have a deep embedding of rules but a shallow embedding of variables, which means that where we set out the text of a "rule", Isabelle interprets this as all instances of (the variables in) that rule. We define a rule as a data structure, a pair of a list of premises and a conclusion, and Isabelle provides the infinitely many substitution instances of these rules.

3.1 Formalising Formulae, Sequents and Sequent Rules

We first encode the grammar for recognising formulae as below:

```
datatype formula = BImp formula formula ("_ -> _" [61,61] 60)
                 | T
                 | FV string           (* formula variable *)
                 | PP string           (* primitive proposition *)
```

Here, there are four type constructors `BImp`, `T`, `FV` and `PP`. The first two encode the implication connective $\to$ and the verum constant $\mathbf{t}$ while the second two encode formula variables such as A and primitive propositions (atomic formulae) such as p and q. The constructor `BImp` takes two formulae as arguments while `FV` and `PP` each take one string argument which is simply the string we want to use for that variable or atomic formula. The code at the end of the first line declares `->` as an alternative symbol for `BImp`. For example, `BImp (FV "A") (PP "q")` encodes $A \to q$, but it can also be written as `(FV "A") -> (PP "q")`.

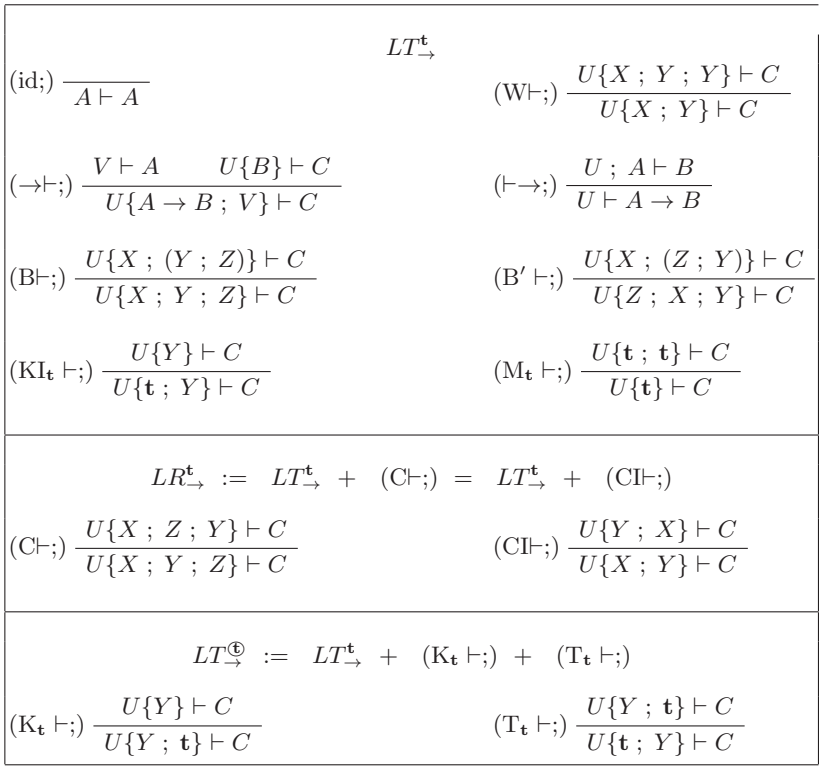

Fig. 3. Various Consecution Rules

Structures are encoded using a parameter `'f` as a type variable:

```
datatype 'f structr = Sf 'f
            | SemiC "'f structr" "'f structr" ("_; _" [20,21] 20)
```

Thus `Sf f` forms an atomic structure from a formula `f`, while `SemiC s1 s2` forms a binary structure from two substructures. A shorthand notation for `SemiC` allows us to write `((FV "A" -> FV "B") ; (PP "q"))` for $((A \rightarrow B)\ ;\ q)$.

A sequent is encoded using two parameters `'l` and `'r` as type variables:

```
datatype ('l, 'r) sequent =
          Sequent "'l" "'r"                ("((_)/ |- (_))" [6,6] 5)
```

An alternative is to replace the prefix `Sequent` with an infix `|-`. So the sequent $A, B \vdash C$ is represented as `Sequent {A,B} C` or as `{A,B} |- C`. The HOL expression `formula multiset` captures the type of formula multisets.

A rule type `psc` is represented as a pair consisting of a list of premises and a conclusion over some parametric type using the type variable `'a`:

```
types 'a psc = "'a list * 'a"
```

Using it, we can define the $(\rightarrow\vdash)$ rule as below:

```
consts impL :: "(formula multiset, formula) sequent psc set"
inductive "impL"
  intrs I "([alpha |- A,                    mins B beta |- C],
               mins (A -> B) alpha + beta |- C) : impL"
```

Here, we first declare that `impL` accepts only sequents built from an antecedent multiset and a single formula succedent: thus `'a` must be `((formula multiset, formula) sequent)`. The function `mins` stands for "multiset insert". The function `+` returns the multiset union of its two arguments, where the number of occurrences of each item is the sum of the number of occurrences in the two multisets. So `mins B alpha + beta` forces `alpha` and `beta` to be of type multiset, and returns the result of inserting one occurrence of `B` into their multiset-union. The sequents `alpha |- A` and `mins B beta |- C` are separated by a comma and enclosed in `[` and `]` to create a list as the first component of the pair formed using `(_ , _)`. The conclusion `mins B alpha + beta |- C` is the second component of this pair, thus forming a rule. The word `inductive` declares `impL` as the smallest set constructed from such pairs (using all possible values for `A`, `B`, `C`, `alpha` and `beta`), which also explains the final `set` in its type declaration.

We now explain our encoding of the "square bracket" conditions in rule $[\rightarrow\vdash]$:

```
consts sqbr :: "'a => 'a multiset => 'a multiset set"
inductive "sqbr dist fmls"
  intrs I "cms <= mins dist fmls ==>
           set_of (mins dist fmls) = set_of cms ==>
           ALL fml. count fmls fml <= Suc (count cms fml)
                                    ==>  cms : sqbr dist fmls"
```

Here, `sqbr` accepts two arguments: `dist` of type `'a` and `fmls` of multisets over type `'a`. It returns a set of multisets over type `'a`. The line beginning `intrs` declares that the conclusion multiset `cms` is a submultiset of the multiset obtained by inserting one occurrence of the distinguished formula `dist` into the multiset `fmls`. The next line declares that, as sets, the objects `cms` and `mins dist fmls` are identical: they differ only in the number of occurrences of some formulae in them, including `dist`. The third line declares that if n is the number of occurrences of any formula `fml` in `fmls` and m is the number of occurrences of `fml` in `cms` then $n \leq m + 1$: in other words, $n - 1 \leq m \leq n$ if `fml` $\neq$ `dist` else $n - 1 \leq m \leq n + 1$ if `fml` $=$ `dist`. The rule $[\rightarrow\vdash]$ is then encodes as `sbimpL`:

```
inductive "sbimpL" intrs
     I "cms : sqbr (A -> B) (alpha + beta) ==>
              ([alpha |- A, mins B beta |- C], cms |- C) : sbimpL"
```

Thus premises `alpha |- A` and `mins B beta |- C` and conclusion `cms |- C` must obey the definition of `sbimpL`, where `A -> B` is `dist` and `alpha + beta` is `fmls`, given that `cms` is some possible set in `sqbr (A -> B) (alpha + beta)`.

We define the other rules in a similar way, giving rise to our various calculi. Here is the definition of the calculus $LR_{\rightarrow}$.

Definition 1 (LRi). *The rule instance* `psc` *is in the sequent calculus* `LRi` *if it is an instance of any of the rules* `iid_rls`*,* `impL`*,* `impR` *and* `lctr_rls`*:*

```
inductive "LRi"
intrs
    id   "psc : iid_rls  ==> psc  : LRi"
    impL "psc : impL     ==> psc  : LRi"
    impR "psc : impR     ==> psc  : LRi"
    W    "psc : lctr_rls ==> psc  : LRi"
```

Here, `LRi` is the smallest set of instances of premises-conclusion pairs that obeys the four clauses `id`, `impK`, `impR`, and `W`. Each clause checks whether a premises-conclusion pair is an instance of some rule: for example `impL`. If so, then it adds that premises-conclusion pair to the set of instances in `LRi`.

Formalising structures, consecutions and consecution rules is similar, except that our basic types are structures, rather than multisets, built from formulae.

3.2 Derivability Predicates `derrec` and `derl`

We also use some general functions to describe derivability. An inference rule of type `'a psc` is a list `ps` of premises and a conclusion `c`. Then `derl rls` is the set of rules derivable from the rule set `rls` while `derrec rls prems` is the set of sequents derivable using rules `rls` from the set `prems` of premises. The special case `derrec rls {}` when `prems` is the empty set `{}` captures the set of `rls`-derivable end-sequents. We defined these functions using Isabelle's package for inductively defined sets, and a more detailed expository account of these, with many useful lemmas, is given elsewhere [Gor09].

```
derl      :: "'a psc set => 'a psc set"
derrec    :: "'a psc set => 'a set => 'a set"
```

3.3 Inductive Multi-cut Admissibility via `gen_step2`

Suppose the conclusions `cl` and `cr` have respective derivations as shown below:

$$\dfrac{\dfrac{\texttt{pl}_1 \ \cdots \ \texttt{pl}_n}{\texttt{cl}}\,\rho_l \qquad \dfrac{\texttt{pr}_1 \ \cdots \ \texttt{pr}_m}{\texttt{cr}}\,\rho_r}{?}\,(\mathit{cut}\ ?)$$

The bottom-most rules of the respective derivations are the rules ρ_l and ρ_r with respective premises $\texttt{psl} = [\texttt{pl}_1, \cdots, \texttt{pl}_\texttt{n}]$ and $\texttt{psr} = [\texttt{pr}_1, \cdots, \texttt{pr}_\texttt{m}]$. Since some premises may be identical, the constructs `set psl` and `set psr` return the sets of premises formed from the respective lists. Suppose now that we want to prove an arbitrary property `P` of these derivations, such as (multi)cut-admissibility for a cut-formula `A`. In previous work, we have shown how to generalise cut-admissibility proofs using a predicate called `gen_step2sr` [DG10]. Here we use a slight variant of this principle which we call `gen_step2` as described next.

Definition 2 (gen_step2). *For property P, formula A, a subformula relation sub, two sets of sequents dls and drs, inference rules (psl, cl) and (psr, cr), the property gen_step2 holds iff P A (cl, cr) holds whenever all of the following hold: P A' (dl, dr) holds for all subformulae A' of A and all sequents dl in dls and dr in drs; for each pl ∈ psl, pl ∈ dls and P A (pl, cr) holds; for each pr ∈ psr, pr ∈ drs and P A (cl, pr) holds; cl ∈ dls and cr ∈ drs.*

```
gen_step2 ?P ?A ?sub (dls, drs) ((psl, cl), (psr, cr)) =
   (ALL A'. (A', ?A) : ?sub -->
     (ALL dl:dls. ALL dr:drs. ?P A' (dl, dr))) -->
      (ALL pl:set psl. pl : dls & ?P ?A (pl, cr)) -->
       (ALL pr:set psr. pr : drs & ?P ?A (cl, pr)) -->
                     cl : dls --> cr : drs --> ?P ?A (cl, cr))
```

Given two sequents `cl` and `cr`, suppose we want `P A cl cr` to capture cut-admissibility of a particular cut-formula `A`. By letting `dls` and `drs` be the set of derivable sequents, the definition of `gen_step2` captures that we can assume:

(a) cut admissibility holds in respect of a smaller cut-formula `A'`
(b) cut admissibility holds between the sequent `cr` on the right and the preceding sequents `psl` in the derivation on the left
(c) cut admissibility holds between the sequent `cl` on the left and the preceding sequents `psr` in the derivation on the right.

The main theorem `gen_step2_lem` below for proving an arbitrary property P states that if `seqa` and `seqb` are derivable, and `gen_step2 P` holds generally, then `P A` holds between `seqa` and `seqb`. In this theorem, the constructions `derrec ?rlsa` {} and `derrec ?rlsb` {} are respectively the set of sequents recursively derivable from the empty set {} of premises using the rule sets `rlsa` and `rlsb`, which potentially could be different rule sets, but are both the same in our case.

Theorem 1 (gen_step2_lem). *An arbitrary property P holds of an arbitrary formula B, and a pair of arbitrary sequents seqa and seqb if: B is in the well-founded part of the subformula relation; sequent seqa is rlsa-derivable; sequent seqb is rlsb-derivable; and for all formulae A, and all rlsa-rules (psl, cl) and rlsb-rules (psr, cr), our induction step condition gen_step2 ?P A ?sub (derrec ?rlsa {}, derrec ?rlsb {}) ((psa, ca), (psb, cb)) holds:*

```
[| ?B : wfp ?sub ;
   ?seqa : derrec ?rlsa {} ; ?seqb : derrec ?rlsb {} ;
   ALL A. ALL (psa, ca):?rlsa. ALL (psb, cb):?rlsb.
          gen_step2 ?P A ?sub (derrec ?rlsa {}, derrec ?rlsb {})
               ((psa, ca), (psb, cb)) |]  ==> ?P ?B (?seqa, ?seqb)
```

Next we define the general property `P` to be that the sequent that results from multi-cutting `cl` and `cr` on cut-formula `A` is `rls`-derivable.

Definition 3 (mcd rls). *The predicate mcd ?rls ?A (?cl, ?cr) means that the conclusion $Xl, Xr \vdash B$ of a multicut-instance is recursively derivable from the empty set of premises using rule set rls if* `cl` $= X_l \vdash A$ *and* `cr` $= X_r, A^n \vdash B$, *where $n > 0$, are the left and right premises, respectively, of the multicut:*

```
mcd ?rls ?A (?cl, ?cr) = (ALL Xl Xr n B.
     ?cl = (Xl |- ?A) & ?cr = (Xr + times (Suc n) {#?A#} |- B)
                          --> (Xl + Xr |- B) : derrec ?rls {})
```

Multicut admissibility is `mca`, which requires that `cl` and `cr` are derivable.

Definition 4 (`mca`). *For any rule set* `rls`*, any formula* `A`*, and any sequents* `cl` *and* `cr`*, the predicate* `mca rls A (cl, ?cr)` *means: if* `cl` *and* `cr` *are* `rls`*-derivable then* `mcd rls A (cl, cr)` *holds:*

```
mca ?rls ?A (?cl, ?cr) = (?cl : derrec ?rls {} -->
         ?cr : derrec ?rls {} --> mcd ?rls ?A (?cl, ?cr))
```

Using multicut instead of cut avoids the difficulty caused by the contraction rule.

3.4 Modular Multicut Instances

The file `LRica.ML` is relevant here. Bimbó and Dunn [BD12] begin with the sequent calculus $LR_{\to}$ and its slight extension $LR^t_{\to}$. Now when admissibility of cut, or of any other rule, holds of a calculus, it does not necessarily hold in a larger calculus. But each proof-step in cut-admissibility for $LR_{\to}$ is mimicked in $LR^t_{\to}$, requiring extra steps only for the extra rules contained in $LR^t_{\to}$.

From Theorem 1, proving cut-admissibility requires proving `gen_step2 (mcd rls)` for each possibility of the last rules used to derive the premises of the proposed cut. We now show how to express these results in a way which allows them to be used for any containing logic. We refer to the diagram above Definition 2.

Lemma 1 (`gsm_impR_R`). *If the rule set* `rls` *contains* $(\vdash\to)$*, and the rule* ρ_r *on the right is an instance of the* $(\vdash\to)$ *rule, then* `gen_step2 (mcd rls)` *holds:*

```
impR <= ?rls
    ==> gen_step2 (mcd ?rls) ?A ?any (?drsl, derrec ?rls {})
     ((?psl, ?cl), ([mins ?G ?alpha |- ?H], ?alpha |- ?G -> ?H))
```

Notice that it does not matter how the left premise `cl` is derived, just that (as contained in the definition of `gen_step2`) cut-admissibility (in the sense of `mcd`, not `mca`), holds between it and the premises `psr` of the final rule ρ_r on the right. The term `?drsl` is `derrec ?rls {}` in this proof: see Definition 2.

The form of the theorem indicates which part of the inductive hypothesis is used: for example, the third argument of `gen_step` is either `?any` or `ipsubfml` depending on whether or not cut-admissibility for subformulae is needed.

4 Various Machine-checked Results

The Calculi $LR_{\to}$ and $LR^{\mathbf{t}}_{\to}$.

Definition 5 (`LRit`). *A rule instance* `psc` *is in the calculus* `LRit` *($LR^{\mathbf{t}}_{\to}$) if it is in the calculus* `LRi` *($LR_{\to}$) or is an instance of the rule* $(\mathbf{t}\vdash)$ *or* $(\vdash \mathbf{t})$*:*

```
inductive "LRit"
intrs
    LRi "psc : LRi ==> psc : LRit"
    tL  "psc : tL  ==> psc : LRit"
    tR  "psc : tR  ==> psc : LRit"
```

Theorem 2 (mca_LRi). *The sequent calculus $LR_{\to}$ enjoys multi-cut admissibility:* `mca LRi ?A (?cl, ?cr)`.

Theorem 3 (mca_LRit). *The calculus* `LRit` *enjoys multicut-admissibility:* `mca LRit ?A (?cl, ?cr)`.

Corollary 1 (Theorem 2.2 of [BD12]). *The single-cut rule is admissible in $LR_{\to}$ and $LR^{\mathbf{t}}_{\to}$: if $\Gamma_1 \vdash A$ and $\Gamma_2, A \vdash C$ are derivable then so is $\Gamma_1, \Gamma_2 \vdash C$.*

The Calculi $[LR_{\to}]$ and $[LR^{\mathbf{t}}_{\to}]$. The file `LRisbcca.ML` is relevant here.

These calculi modify $LR_{\to}$ and $LR^{t}_{\to}$ by deleting the contraction rule $(W\vdash)$, but modifying the $(\to\vdash)$ rule into a new rule called $[\to\vdash]$ that allows a limited amount of contraction. Bimbó and Dunn [BD12, Theorem 2.4] state that the cut rule is admissible, by a proof similar to that for $LR^{\mathbf{t}}_{\to}$ [BD12, Theorem 2.2]. We were unable to prove the result in this way but we were able to prove contraction-admissibility instead using a technique similar to that for cut-admissibility, but simpler, as it is a property of one sequent, not two. Again, there are two versions.

Definition 6 (lcd). *For any rule set* `rls`*, and any formula* `A`*, and any sequent* `c`*, the predicate* `lcd rls A c` *means: for all multisets* `X` *and all formulae* `B`*, if* `c` *is $X, A, A \vdash B$ then the sequent $X, A \vdash B$ is* `rls`*-derivable.*

```
lcd ?rls ?A ?c == ALL X B. ?c =
(X + {#?A#} + {#?A#} |- B) --> (X + {#?A#} |- B) : derrec ?rls {}
```

Definition 7 (lca). *For any rule set* `rls`*, and any formula* `A`*, and any sequent* `c`*, the predicate* `lca rls A c` *means: for all multisets* `X` *and all formulae* `B`*, if* `c` *is* `rls`*-derivable then* `c` *enjoys* `lcd rls A c`*:*

```
lca ?rls ?A ?c == ?c : derrec ?rls {} --> lcd ?rls ?A ?c
```

Definition 8 (LRisb **and** LRitsb). *The rules of the sequent calculus* `LRisb` *(resp.* `LRitsb`*) are those of* `LRi`*, Def 1 (resp.* `LRit`*, Def 5) omitting the $((W\vdash))$ rule, and changing the $(\to\vdash)$ rule to the rule $[\to\vdash]$ (see Fig 2)*

```
inductive "LRisb"
intrs id     "psc : iid_rls ==> psc : LRisb"
      sbimpL "psc : sbimpL  ==> psc : LRisb"
      impR   "psc : impR    ==> psc : LRisb"

inductive "LRitsb"
intrs LRisb "psc : LRisb ==> psc : LRitsb"
      tL    "psc : tL    ==> psc : LRitsb"
      tR    "psc : tR    ==> psc : LRitsb"
```

Theorem 4 (`lca_LRisb` and `lca_LRitsb`). *The contraction rule is admissible in the calculi* ***LRisb*** *and* ***LRitsb****:* `lca LRisb ?A ?c` *and* `lca LRitsb ?A ?c`*.*

Having proved contraction admissibility for `LRisb` and `LRitsb`, we prove their equivalence to `LRi` and `LRit` respectively as follows.

Theorem 5 (`LRi_LRisb`, `LRisb_LRi`, `LRisb_eqv_LRi`). *Each rule from* ***LRi/LRisb*** *is admissible/derivable in* ***LRisb/LRi****. So* ***LRi*** *and* ***LRisb*** *are equivalent.*

Theorem 6 (`LRit_LRitsb`, `LRitsb_LRit`). *Each rule from* ***LRit/LRitsb*** *is admissible/derivable in* ***LRitsb/LRit****, so* ***LRit*** *and* ***LRitsb*** *are equivalent.*

These give part of [BD12, Lemma 2.5] and give [BD12, Lemma 2.4].

Theorem 7 (`mca_LRisb` and `mca_LRitsb`). *Both* ***LRisb*** *and* ***LRitsb*** *enjoy multicut-admissibility:* `mca LRisb ?A (?cl, ?cr)` *and* `mca LRitsb ?A (?cl, ?cr)`*.*

Corollary 2 (Kripke 1959). *The single-cut rule is admissible in* $[LR_{\rightarrow}]$ *and* $[LR^{\mathsf{t}}_{\rightarrow}]$*: if* $\Gamma_1 \vdash A$ *and* $\Gamma_2, A \vdash C$ *are derivable then so is* $\Gamma_1, \Gamma_2 \vdash C$*.*

The Calculi $LT_{\rightarrow}$ and $LT^{\textcircled{t}}_{\rightarrow}$. The file `LTitca.ML` is relevant here. We now need to encode structures with a hole and encode consecutions and rules built from consecutions where the action happens at the hole. We have explained how to achieve this for nested sequent calculi elsewhere [DCGT14] and so the sequel is rather terse. The main point here is that all the action happens in the antecedent and so we concentrate on the relation holding between such contexts.

Definition 9 (`sctxt`). *If* ***(a, b) ∈ r*** *then* ***(a, b) ∈ sctxt r****. Every* ***(a, b) ∈ sctxt r*** *can be extended by prefixing/postfixing with an arbitrary context* ***C****.*

```
consts sctxt :: "'f structr relation trf"
(* closure of rule structure relation under context *)
inductive "sctxt r" intrs
    scid "(a, b): r ==> (a, b) : sctxt r"
    scL  "(a, b): sctxt r ==> (C;a, C;b) : sctxt r"
    scR  "(a, b): sctxt r ==> (a;C, b;C) : sctxt r"
```

A structure with a hole (a context) is turned into a consecution by simply adding a turnstile and a singleton on the right as follows. The relation between the antecedents is also retained.

Definition 10 (`lctxt`). *The set* ***lctxt r*** *is the smallest set of rule instances obtained by extending every pair* ***(As, Bs) ∈ sctxt r*** *into the rule instance* ***([As |- E], Bs |- E)*** *with premise* ***As |- E*** *and conclusion* ***Bs |- E****.*

```
consts lctxt ::
    "'f structr relation => ('f structr, 'f) sequent psc set"
inductive "lctxt r" intrs
    I "(As, Bs) : sctxt r ==> ([As |- E], Bs |- E) : lctxt r"
```

We define `LTit_lc` as the pairs (X, Y) giving us rules of the form at right. Then `lctxt LTit_lc` is the set of such deep structural rules in `LTit`. First, `LTit_lcsub` are the pairs (X, Y) which form rules of the form at right where X and Y consist only of substitutable structure variables (ie unlike the rules involving $\mathbf{t}$).

$$\frac{U\{X\} \vdash C}{U\{Y\} \vdash C}$$

Definition 11. *`LTit_lcsub` is the smallest set of left-context action (pairs of structural transformations) instances of the (combinator) permutations below.*

```
inductive "LTit_lcsub" (* fully substitutable rules *)
intrs B  "psc : lcB ==> psc : LTit_lcsub"
      Bd "psc : lcBd ==> psc : LTit_lcsub"
      W  "psc : lcW ==> psc : LTit_lcsub"
```

Here, we define `lcB`, `lcBd` and `lcW` to give us the rules (B$\vdash$;), (B$'\vdash$;) and (W$\vdash$;). Next, we define the separate relation `lcC` to give ($C \vdash$;) similarly:

```
inductive "lcB"  intrs I "(Bs; (Cs; Ds), Bs; Cs; Ds) : lcB"
inductive "lcBd" intrs I "(Bs; (Cs; Ds), Cs; Bs; Ds) : lcBd"
inductive "lcW"  intrs I "(Bs; Cs; Cs, Bs; Cs)       : lcW"
inductive "lcC"  intrs I "(Bs; Cs; Ds, Bs; Ds; Cs)   : lcC"
```

Here, we elide parentheses by associating to the left and writing `(Bs; (Cs; Ds), Bs; Cs; Ds)` instead of `((Bs; (Cs; Ds)), (Bs; Cs; Ds))`. So we now have the permutations that correspond to the actions that happen at the hole. We now need to turn these actions into rules formed from consecutions.

Definition 12 (`LTit_lc`). *`LTit_lc` is the smallest set of rule instances `psc` formed by extending hole permutation pairs into consecution rules*

```
inductive "LTit_lc"
  intrs sub "psc : LTit_lcsub ==> psc : LTit_lc"
        KIt "psc : KIt T ==> psc : LTit_lc"
        Mt  "(Sf T; Sf T, Sf T) : LTit_lc"
inductive "KIt fml" intrs I "(Bs, Sf fml; Bs) : KIt fml"
```

Here, the construction `Sf T` casts the formula `T` into an atomic structure.

We now need to turn these pairs into proper rules built out of consecutions and also add the usual logical rules.

Definition 13 (`LTit`). *`LTit` is the smallest set of rule instances `psc` which are instances of the logical rules `lcid`, `lcimpR`, `lcimpL`, and of the structural rules corresponding to the combinator permutations `B`, `Bd` and `W`:*

```
inductive "LTit" intrs
    id      "psc : lcid ==> psc : LTit"
    impR    "psc : lcimpR ==> psc : LTit"
    impL    "psc : lcimpL ==> psc : LTit"
    lcrules "psc : lctxt LTit_lc ==> psc : LTit"
```

Here, we have omitted the definitions of the consecution rules `lcid`, `lcimpR`, `lcimpL`. Similar definitions to `LTit` allows us to compose the rule sets `LRitsc`, and `LTitc` (omitted) corresponding to the consecution calculi $LR^{\mathbf{t}}_{\rightarrow}$ and $LT^{\textcircled{\mathbf{t}}}_{\rightarrow}$.

4.1 A Structural Analogue of Multicut

Since these calculi contain a contraction rule we prefer to show admissibility of multicut rather than cut. Following Dunn [Dun73], given premise sequents $X \vdash A$ and $Y \vdash B$, we consider the "multicut" that replaces each one of n (rather than all) occurrences of A in Y by an X, to give Z (say):

$$\text{(multicut)}\ \frac{X \vdash A \qquad Y\{A\}\{A\}\cdots\{A\} \vdash B}{Y\{X\}\{X\}\cdots\{X\} \vdash B}$$

The relationship between Y and Z described above, is encoded as `strrep`.

Definition 14 (`strrep`)**.**

```
consts strrep :: "'f structr pair set => 'f structr pair set"
inductive "strrep S" intrs
    same "(s, s) : strrep S"
    repl "p : S ==> p : strrep S"
    sc   "(u, v) : strrep S ==>
          (x, y) : strrep S ==> (u; x, v; y) : strrep S"
```

This introduces the issue that where $P\{A\}$ and $C\{A\}$ are (say) the antecedents of the premise and conclusion of a rule, and $(P\{A\}, C\{A\}) \in$ `sctxt r` for a relation (set of pairs) `r`, eg `r = {(B;(C;D), B;C;D)}` (for the $(B \vdash;)$ rule), and multicutting with $X \vdash A$ would give $C\{X\}$, ie $(C\{A\}, C\{X\}) \in$ `strrep {(Sf A, X)}`, then we need to "close the box" with $P\{X\}$, where $(P\{A\}, P\{X\}) \in$ `strrep {(Sf A, X)}` and $(P\{X\}, C\{X\}) \in$ `sctxt r`. The easiest instance is where `r` is a set of pairs which are entirely substitutable: for example the pair `{(B;(C;D), B;C;D)}` for $(B \vdash;)$, rather than the pair `{A, (t;A)}` for $(KI_{\mathbf{t}} \vdash;)$.

Lemma 2 (`strrep_sctxt_lcsub`)**.**

```
[| (?PA, ?CA) : sctxt LTit_lcsub ;
   (?CA, ?CX) : strrep {(Sf ?A, ?X)} |] ==>
                          EX PX. (?PA, PX) : strrep {(Sf ?A, ?X)}
                               & (PX, ?CX) : sctxt LTit_lcsub
```

Here `LTit_lcsub` from Definition 13 is the set of pairs of the form found in the rules $(B \vdash;)$, $(B' \vdash;)$, $(W \vdash;)$ from $LT^{\mathbf{t}}_{\rightarrow}$.

For the verum constant `T`, the corresponding result is (for example):

Lemma 3 (`strrep_sctxt_KIt`)**.**

```
[| (?PA, ?CA) : sctxt (KIt T); ?A ~= T;
         (?CA, ?CX) : strrep {(Sf ?A, ?X)} |] ==> EX PX.
  (?PA, PX) : strrep {(Sf ?A, ?X)} & (PX, ?CX) : sctxt (KIt T)
```

So the multicut-admissibility property we prove inductively is `mclcd`.

Definition 15 (`mclcd`)**.** *The predicate* `mclcd` *means: if* `cl` $= Xl \vdash A$ *and* `cr` $= Xr \vdash B$ *and* Y *is obtained from* Xr *by replacing some instances of* A *by* Xl*, then* $Y \vdash B$ *is* `rls`*-derivable.*

```
mclcd ?rls ?A (?cl, ?cr) =
 (ALL Xl Xr Y B. ?cl = (Xl |- ?A) --> ?cr = (Xr |- B) -->
  (Xr, Y) : strrep {(Sf ?A, Xl)} --> (Y |- B) : derrec ?rls {})
```

The version conditional on `cl` and `cr` being derivable is

Definition 16 (`mclca`). *The predicate* `mclca` *says that if* `cl` $= Xl \vdash A$ *and* `cr` $= Xr \vdash B$ *are* `rls`*-derivable, and* Y *is obtained from* Xr *by replacing some instances of* A *by* Xl*, then* $Y \vdash B$ *is* `rls`*-derivable.*

```
mclca ?rls ?A (?cl, ?cr) = (?cl : derrec ?rls {} -->
               ?cr : derrec ?rls {} --> mclcd ?rls ?A (?cl, ?cr))
```

4.2 Results for Consecution Calculi

The next result is an example of many results (omitted) expressed to apply to a rule set which is a superset of a given set. Thus it and the omitted results are useful for all of the consecution calculi $LT^{\mathbf{t}}_{\rightarrow}$, $LR^{\mathbf{t}}_{\rightarrow}$ and $LT^{\textcircled{t}}_{\rightarrow}$. In fact we combined all these results to get

Lemma 4 (`gsmcl_LTit`). *If* `rls` *contains* `LTit` *and rules* `(psl, cl)` *and* `(psr, cr)` *are from* `LTit` *then* `gen_step2 (mclcd rls)` *holds:*

```
[| LTit <= ?rls ; (?psl, ?cl) : LTit ; (?psr, ?cr) : LTit |]
   ==> gen_step2 (mclcd ?rls) ?A ipsubfml
       (derrec ?rls {}, derrec ?rls {}) ((?psl, ?cl), ?psr, ?cr)
```

Theorem 8 (`mclca_LTit`). *The consecution calculus* $LT^{\mathbf{t}}_{\rightarrow}$ *enjoys multi-cut admissibility: if the consecution* $V \vdash A$ *and the consecution* $U\{A\}\{A\}\cdots\{A\} \vdash C$ *are derivable then the consecution* $U\{V\}\{V\}\cdots\{V\} \vdash C$ *is derivable.*
`mclca LTit ?A (?cl, ?cr)`.

We obtain the single-cut admissibility result for $LT^{\mathbf{t}}_{\rightarrow}$ which is only asserted by Bimbó and Dunn [BD12, line 10, pg 500] since it is proved elsewhere.

Corollary 3 (Bimbó and Dunn line 10, pg 500 [BD12]). *The single-cut rule is admissible in* $LT^{\mathbf{t}}_{\rightarrow}$*: if the consecutions* $V \vdash A$ *and* $U\{A\} \vdash C$ *are derivable then so is the consecution* $U\{V\} \vdash C$*.*

Extending the proof to the other calculi was quite easy since we only needed to deal with the cases involving a few additional rules on either side.

Theorem 9 (`mclca_LTitc` **and** `mclca_LRitsc`). *The consecution calculi* $LT^{\textcircled{t}}_{\rightarrow}$ *and* $LR^{\mathbf{t}}_{\rightarrow\ ;}$ *enjoy multi-cut admissibility: if the consecution* $V \vdash A$ *and consecution* $U\{A\}\{A\}\cdots\{A\} \vdash C$ *are derivable then so is* $U\{V\}\{V\}\cdots\{V\} \vdash C$*.*

```
mclca LTitc ?A (?cl, ?cr)
mclca LRitsc ?A (?cl, ?cr)
```

Corollary 4 (Bimbó and Dunn Thm 3.2 and Thm 5.2 [BD12]). *The single-cut rule is admissible in* $LR^{\mathbf{t}}_{\rightarrow\ ;}$ *and* $LT^{\textcircled{t}}_{\rightarrow}$*: if the consecutions* $V \vdash A$ *and* $U\{A\} \vdash C$ *are derivable then so is the consecution* $U\{V\} \vdash C$*.*

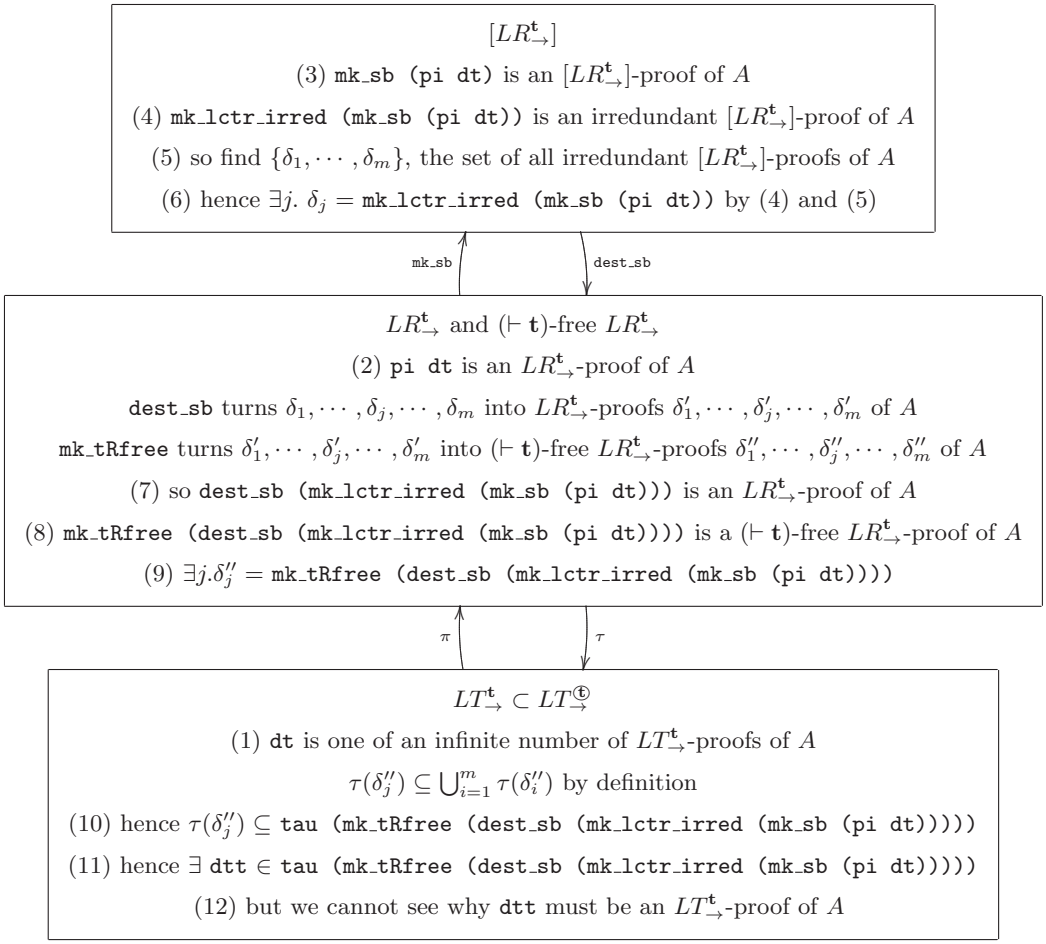

Fig. 4. Proof Plan

5 A Proof Plan of The Crucial Lemma 11

A putative constructive proof plan of [BD13, Lemma 11] is in Figure 4.

Assertion 1 *If there is an $LT^{\mathbf{t}}_{\rightarrow}$-proof* `dt` *of A then there exists an $[LR^{\mathbf{t}}_{\rightarrow}]$-proof* `dtsb` *of A and there exists an $LT^{\mathbf{t}}_{\rightarrow}$-proof* `dtt` *of A.*

Proof Plan: We start with (1) some given $LT^{\mathbf{t}}_{\rightarrow}$-proof `dt` of A. Applying π gives us (2) there is some $LR^{\mathbf{t}}_{\rightarrow}$-proof `pi dt` of A. By completeness of $[LR^{\mathbf{t}}_{\rightarrow}]$, (3) there is an $[LR^{\mathbf{t}}_{\rightarrow}]$-proof `mk_sb (pi dt)` of A. Then, (4) there must be an irredundant such $[LR^{\mathbf{t}}_{\rightarrow}]$-proof `mk_lctr_irred (mk_sb (pi dt))` of A. Then (7) the function `dest_sb` transforms an $[LR^{\mathbf{t}}_{\rightarrow}]$-proof into an $LR^{\mathbf{t}}_{\rightarrow}$-proof by replacing the rule

$[\to\vdash]$ with an instance of the $(\to\vdash)$ rule followed by the appropriate number of explicit applications of the contraction rule $(W\vdash)$ thereby "destroying the square brackets". Then (8) the function `mk_tRfree` transforms the resulting $LR^{\mathbf{t}}_{\to}$-proof into a $(\vdash \mathbf{t})$-free $LR^{\mathbf{t}}_{\to}$-proof. Finally, `tau` transforms this $(\vdash \mathbf{t})$-free $LR^{\mathbf{t}}_{\to}$-proof into possibly many $LT^{\textcircled{t}}_{\to}$ proofs, hence (11) there must exist some $LT^{\textcircled{t}}_{\to}$-proof `dtt` of A. But (12) why should (any such) `dtt` be an $LT^{\mathbf{t}}_{\to}$-proof of A?

So, how to complete our proof plan? The first point is that the existence of `dt` must lead to a `dtt`. There are two plausible approaches:

(a) `dtt` is equal, similar or somehow related to `dt`
(b) τ must produce enough proofs to guarantee that if there is an $LT^{\mathbf{t}}_{\to}$-proof for A, then τ will give us one. To guarantee this, we may have to apply τ to all $[LR^{\mathbf{t}}_{\to}]$-proofs of A (after applying `dest_sb` and `mk_tRfree` to them).

The discussion by Bimbó and Dunn regarding π mentioned in the last two lines of the proof of their Lemma 11 seems only relevant to (a) above. The only relevance of π can be for an argument along the lines shown in Figure 4.

However, the proof of Lemma 11 of Bimbó and Dunn does not start with a given $LT^{\mathbf{t}}_{\to}$-proof `dt` of A, but appears to follow option (b) outlined above. That is, it uses their Lemma 5 to deduce that A being a theorem of $T^{\mathbf{t}}_{\to}$ implies the existence of some $[LR^{\mathbf{t}}_{\to}]$-proof of A. So it starts by (5) finding **all** irredundant $[LR^{\mathbf{t}}_{\to}]$-proofs of A, transforming them into $LR^{\mathbf{t}}_{\to}$-proofs by simply making contractions explicit, then transforming them to remove all applications of the $(\vdash \mathbf{t})$ rule, and turning the resulting proofs into $LT^{\textcircled{t}}_{\to}$-proofs by applying τ. Their argument that one of these must be an $LT^{\mathbf{t}}_{\to}$-proof requires considering "all permutations" of the structures involved. But this whole procedure starts with only those proofs which have the contractions allowed by the "square bracket" calculi. Moreover, the decision procedure starts at point (5), but the proof of its completeness starts at point (1), at the hypothesis that A is a theorem of $T^{\mathbf{t}}_{\to}$.

Our proof plan and their proof align if at (5) we find all irredundant $[LR^{\mathbf{t}}_{\to}]$-proofs of A, and then (6) one of them, say δ_j, must be the one we are focusing on. So (8) the result of applying both `dest_sb` and `mk_tRfree` to δ'_j is a $(\vdash \mathbf{t})$-free $LR^{\mathbf{t}}_{\to}$-proof δ''_j of A and (9) δ''_j must be one of the proofs obtained by doing these transformations to all of the proofs from (5). Finally, τ transforms any one of these $(\vdash \mathbf{t})$-free $LR^{\mathbf{t}}_{\to}$-proofs into possibly many proofs and hence `dtt` is in $\tau(\delta''_j)$. But again, (12), we cannot see why this final proof has to be an $LT^{\mathbf{t}}_{\to}$-proof.

Indeed (12), and our proof plan would hold if we could prove that `dt = dtt`.

Assertion 2 (`tau_irr_sb_pi`) *If `dt` is an $LT^{\mathbf{t}}_{\to}$-proof of A then*
`dt` ∈ `tau (mk_tRfree (dest_sb (mk_lctr_irred (mk_sb (pi dt)))))`.

We cannot see why this assertion should hold. Allowing that it would be true that `dt` is in $\tau(\pi($`dt`$))$, we should consider the differences between the proofs $\pi($`dt`$)$ and `mk_tRfree (dest_sb (mk_lctr_irred (mk_sb (pi dt))))`. Now `mk_lctr_irred` excises parts of a proof and uses height-preserving contraction admissibility, so it (probably) simplifies a proof. Also, `mk_tRfree` makes changes which are probably insignificant. But the `dest_sb (... mk_sb ...)` combination moves

contractions around, relative to occurrences of $(\rightarrow\vdash)$, since `mk_sb` must remove contractions that are not immediately below $(\rightarrow\vdash)$, and then make up for their removal by inserting appropriate contractions immediately below the $(\rightarrow\vdash)$ rules. But π alone does not do such movements by [BD13, Lemma 6].

6 Conclusions

We have machine-checked all of the proof-theoretic claims made by Bimbó and Dunn [BD12,BD13] including the three lemmata which are at the heart of the decidability argument [BD13, Lemmata 8,9,10]. However, we were not able to prove them in that order as our proof of Lemma 9 depends upon our proof of Lemma 10. Moreover, we are yet to be convinced of the correctness of Lemma 11 which ensures that no $LT^{\mathbf{t}}_{\rightarrow}$ derivation is lost in the transformations of proofs which correspond to $[LR^{\mathbf{t}}_{\rightarrow}]$-proofs: see Figure 4. Our files are at `http://users.cecs.anu.edu.au/~jeremy/isabelle/2005/bimbo-dunn/` and the URL address `http://users.cecs.anu.edu.au/~jeremy/isabelle/2005/bimbo-dunn/ticket-instructions.html` contains instructions for running them.

Acknowledgements. We are grateful to Katalin Bimbó, Michael Dunn and John Slaney for their helpful comments. All remaining errors are our own.

References

[BD12] Katalin Bimbó and J Michael Dunn. New consecution calculi for $R^{\mathbf{t}}_{\rightarrow}$. *Notre Dame Journal of Formal Logic*, 53(4), 2012.

[BD13] Katalin Bimbó and J Michael Dunn. On the decidability of implicational ticket entailment. *The Journal of Symbolic Logic*, 78(1), March 2013.

[DCGT14] Jeremy E. Dawson, Ranald Clouston, Rajeev Goré, and Alwen Tiu. From display calculi to deep nested sequent calculi: Formalised for full intuitionistic linear logic. In *Proc 8th Theoretical Comp Sci*, pp 250–264, 2014.

[DG02] Jeremy E. Dawson and Rajeev Goré. Formalised cut admissibility for display logic. In *Proc TPHOLs*, pages 131–147, 2002.

[DG10] Jeremy E. Dawson and Rajeev Goré. Generic methods for formalising sequent calculi applied to provability logic. In *Proc LPAR-17*, pages 263–277.

[Dun73] J. M. Dunn. (abstract only) A 'Gentzen system' for positive relevant implication. *Journal of Symbolic Logic*, 38:356–357, 1973.

[Gor09] Rajeev Goré. Machine checking proof theory: An application of logic to logic. In *Proc ICLA*, pages 23–35, 2009.

[GR12] Rajeev Goré and Revantha Ramanayake. Valentini's cut-elimination for provability logic resolved. *Rew. Symb. Logic*, 5(2):212–238, 2012.

[Pad11] Vincent Padovani. Ticket entailment is decidable. *CoRR*, abs/1106.1875, 2011.