Machine-checked Meta-theory of Dual-Tableaux for Intuitionistic Logic

Jeremy E Dawson and Rajeev Goré
Research School of Computer Science
The Australian National University

Abstract

We describe how we formalised the meta-theory of Melvin Fitting's dual-tableaux calculi for intuitionistic logic using the HOL4 interactive theorem prover. The paper is intended for readers familiar with dual-tableaux who might be interested in, but daunted by, the idea of formalising the required notions in a modern interactive theorem prover.

Contents

1 Introduction and motivation

Tableaux calculi originated with the work of Beth in the 1950s [Bet53]. In 1959, they were used by Kripke [Kri59] to prove the soundness and completeness of his semantics for modal logics such as S5. They were extended to many modal logics and to intuitionistic logic by Melvin Fitting [Fit83] in the 1970s. Since 1992, they have gained a life of their own via the conference series named "Theorem Proving with Analytic Tableaux and Related Methods" which had its 25th anniversary in Brazil in 2017. However, an honest appraisal of the literature must acknowledge a parallel, and sometimes more advanced, tradition known as "dual-tableaux" which arose in Poland from the work of Rasiowa, Sikorski and Ewa Orlowska [OGP11]. Here, we pay homage to that tradition.

Over the past decade, we have formalised many aspects of proof-theory including sequent calculi [DG10] and display calculi [DG02]. Here, we turn our attention to the meta-theory of dual-tableaux calculi. Specifically, we show how to formalise the semantic soundness and completeness proofs for the dual-tableaux calculi for intuitionistic logic given by Melvin Fitting [Fit17]. Our hope is that it will serve as a guide to others who may want to follow in our footsteps.

We assume that the reader is familiar with Fitting's chapter [Fit17] in this volume, but not familiar with interactive theorem proving. All of our `HOL4` files can be found here: `http://users.cecs.anu.edu.au/~jeremy/hol/idt/` and are also available on GitHub at `https://github.com/jeremydaw/idt` in the directory `hol` together with a `README.md` file detailing the HOL version used and instructions for compiling and running the proof files.

2 `HOL4`: an interactive theorem prover

We chose to work with the interactive theorem prover called `HOL4` [Gor08], which implements Dana Scott's Logic of Computable Functions [Sco93]. The user must first encode all of the required definitions of the meta-theory into `HOL4` so we provide most of the details of our definitions. The user then inputs a goal which `HOL4` is asked to prove. Typically, `HOL4` reduces that goal into multiple subgoals and expects the user to choose the next step of the proof to perform. `HOL4` accepts the next step only if it can be used in a sound way to reduce the chosen subgoal into further subgoals. Thus `HOL4` also keeps track of the proof process and the current stage of the proof is always visible to the user. Here, we only show our encoding of the various aspects of the meta-theory of dual-tableaux, and state the lemmata and theorems that we proved inside `HOL4`.

2.1 Why should we trust `HOL4`?

As with many other interactive theorem provers, `HOL4` is a trusted system because its code-base is small, around 4000 lines of code, is written in a functional programming language ML, and has been scrutinised by experts in logic and theorem proving over a period of 40 years. Moreover, `HOL4` can produce a proof-script of the final proof which can be checked by other scrutineers or even other interactive theorem provers.

2.2 The syntax of HOL4

The logic implemented by the HOL4 theorem prover is called classical higher-order logic. It is "higher-order" in that functions and predicates are first-class citizens which can be quantified over and which can be passed to each other as arguments. The basic operators of higher-order logic are these:

$\top$	$\bot$	$\wedge$	$\vee$	$\rightarrow$	$\neg$	$\forall$	$\exists$
`T`	`F`	`/\`	`\/`	`==>`	`~`	`!`	`?`

The syntax of HOL4 also provides for writing lists, pairs and various other data structures which we shall illustrate as needed.

The logic is typed and so there is a separate syntax for defining new types from existing base types provided by HOL4 such as `nat` and `int` for the types of natural numbers and integers respectively.

For example, the symbol `#` is the infix type constructor denoting the type of pairs and the symbol `:` is the infix operator for "member of type". Thus, `a : alpha` encodes that "`a` is of type `alpha`". If also `b : beta` then HOL4 can deduce that `(a, b) : alpha # beta` which encodes "`(a, b)` is in the type that consists of pairs of objects from types `alpha` and `beta` respectively". But generally, type constructors are written postfix, so `alpha set` is the type of sets of items of type `alpha`.

3 Capturing the syntax of dual-tableaux

We now describe how we encoded the syntax of dual-tableaux into HOL4. We build the encoding by first encoding the notion of formulae, then sets of (signed) formulae and then the notions of dual-tableau rules, and finally the notion of (closed) dual-tableaux.

3.1 (Unsigned) Formulae

Strings prefixed with an apostrophe (`'`) are treated by HOL4 as type variables. Using such a type variable `'a`, we first define a datatype for formulae where the type `'a` of an atom is a variable left to be chosen later.

Definition 1 (formula)**.** *The formulae of intuitionistic logic are built from an infinite supply of atomic formulae (`Atom 'a` over some base type `'a`) using the connectives $\wedge$ (`And`), $\vee$ (`Or`), $\rightarrow$ (`Imp`), and $\neg$ (`Not`) as usual:*

```
datatype formula = And formula formula
                 | Or formula formula
                 | Imp formula formula
                 | Not formula
                 | Atom 'a ;
```

Thus we get the type `'a formula` where `'a` is a type variable: in text, we use $\alpha, \beta, \ldots$ for type variables.

The effect of this definition is to declare to HOL4 how it can recognise a string as a formula over the type `'a`: for example, the string `Imp (Atom 1) (Atom 2)` would be recognised by HOL4 as a formula built out of atoms of type `num` of natural numbers since the items in the scope of the string `Atom` are all natural numbers.

The strings `And`, `Or`, `Imp`, `Not` and `Atom` are called the constructors of the datatype `formula`. Moreover, these constructors are the only way to construct formulae, hence we can perform induction on the structure of a formula by starting at the atoms and dealing with a case for each connective.

3.2 Signs and signed formulae

In `HOL4`, there are two pre-defined constants `T` and `F` which make up the pre-defined type `bool` whose members are exactly these two constants. By forming a pair `(b,f)` where `b` is of type `bool` and `f` is of type `'a formula`, we shall use these constants as the signs of our signed-formulae. Thus the type `bool # 'a formula` contains the set of all pairs where the first component is one of the "signs" `T` and `F` and the second component is a formula. We then define `sf` as a type abbreviation for such pairs, thereby hiding the specifics of signed formulae.

Definition 2 (signed formula)**.** *A* signed formula *of type* `sf` *is a pair* `(b, f)` *where* `b` *is either the constant (sign)* `T` *or else is the constant (sign)* `F`*, and* `f` *is a formula:*

```
val _ = Parse.type_abbrev ("sf", ``: (bool # 'a formula)``) ;
```

This just means that all occurrences of the type `'a sf` are expanded to the type `bool # 'a formula` making `sf` a type operator with one argument `'a`.

In the sequel, we sometimes use `sf` as the name of a signed-formula and also sometimes use it as the type of signed formula defined above. Sometimes, we also use `'sf` to indicate an arbitrary type variable, which will be instantiated with the type `'a sf` when used in our `HOL4` proofs. We try to explain these uses when they occur.

3.3 Signed-formula sets as unsigned formula set pairs

For a set S of signed formulae, Fitting [Fit17, Definition 5] defines the two sets $S^T = \{T\ X \mid T\ X \in S\}$ and $S^F = \{F\ X \mid F\ X \in S\}$. He also defines the ability to strip the signs and extract only the unsigned formulae via: $S^\circ = \{X \mid T\ X \in S \text{ or } F\ X \in S\}$. A set of signed-formulae can also be seen as a sequent built out of unsigned formulae collecting the F-signed formulae on the left and the T-signed formulae on the right (without their respective signs). For example, the set $\{(F, a), (T, b), (F, c)\}$ of signed-formulae can be seen as the sequent $a, c \vdash b$ which can be represented by a pair $(\{a, c\}, \{b\})$ of sets of (unsigned) formulae. The ability to move to and fro between these two representations is useful later, so we define functions `mk_seq` and `dest_seq` which switch between the two representations. We also illustrate some `HOL4` syntax.

The construct `'f` is a type variable: it will stand for a (unsigned) formula. In `HOL4`, the type α *set* is "syntactic sugar" for $\alpha \to bool$. Whether a particular term x (say) of type α is or is not in the set P of type α `set` is determined by the value `T` or `F` from `bool` which the predicate $P(x)$ takes. Separately, $x \in P$ (`x IN P`) is logically defined to be $P(x)$ (`P x`). Although `x IN P` and $P(x)$ are provably equivalent, and thus synonymous, they are not identical terms.

The function $FST : \gamma \times \delta \to \gamma$ returns the first component of type γ of a pair of type (γ, δ). If we put $\gamma = bool$ then FST is of type $bool \times \delta \to bool$, which is "syntactic de-sugar" for $(bool \times \delta)\ set$ in HOL4. For a term $z : bool \times \delta$, the predicate $FST(z)$ evaluates to true exactly when the first component of z evaluates to T. Thus $FST : (bool \times \delta)\ set$ means the set of all pairs (of the appropriate type) of the form (T, x).

Definition 3. *The functions* `get_ts` *and* `get_fs` *are defined as:*

```
get_ts      :  (bool # 'f) set -> 'f set
get_ts sfs  = (IMAGE SND (sfs INTER FST))
```
$$= \{f : \texttt{'f} \mid (T, f) \in sfs\}$$

```
get_fs      :  (bool # 'f) set -> 'f set
get_fs sfs  = (IMAGE SND (sfs INTER ($~ o FST))
```
$$= \{f : \texttt{'f} \mid (F, f) \in sfs\}$$

Here, the function `get_ts` accepts a set of pairs of type `bool # 'f`, for any type `'f` and returns a set of items of type `'f`. By giving it an argument of type `'a sf set`, we will cause `'f` to become `'a formula`. We therefore use the name `sfs` in the code to stand for the name of a set of signed-formulae.

The construct `INTER` is the `HOL4` symbol for set intersection $\cap$. Since `sfs` will be type `'a sf set`, the construct `(sfs INTER X)` immediately forces `(sfs INTER X)` to take type `'a sf set` for any function `X`. Putting X to be FST forces FST to be of type $\alpha\ sf \to bool$, effectively putting γ in the general type of FST to $bool$ as described above. So `FST` in the context `(sfs INTER FST)` is the set of all signed-formulae where the first component, the sign, is the Boolean value `T`. The constructor `o` stands for "composition" so `($~o FST)` is the "composition" of Boolean negation `~` and FST, so `($~o FST)` effectively "flips" the required first component `T` to be `F`. Thus `(sfs INTER ($~o FST))` is the set of `F`-signed formulae (pairs) from `sfs`. The construct `SND` returns the second component of a pair while `IMAGE f Y` returns the result of applying f to each $y \in Y$. Thus `(IMAGE SND (sfs INTER ($~o FST))` is the set of second components of the set of `F`-signed formulae (pairs) from `sfs`: that is, the formulae `f` that are signed `F` in `sfs`.

Definition 4. *The sets* `mk_seq sfs` *and* `dest_seq (fs, ts)` *are defined as:*

```
mk_seq            :  (bool # 'f) set -> 'f set # 'f set ;
dest_seq          :  'f set # 'f set -> (bool # 'f) set ;

mk_seq sfs        = (get_fs sfs, get_ts sfs)
```
$$= \{(Fs, Ts) \mid Fs = \{f : \texttt{'f} \mid (F, f) \in sfs\} \text{ and } Ts = \{f : \texttt{'f} \mid (T, f) \in sfs\}\ \}$$

```
dest_seq(fs, ts)  = IMAGE($, F) fs UNION IMAGE ($, T) ts
```
$$= \{\ (F, f) \ \mid\ f \in fs\ \} \cup \{\ (T, f)\ \mid\ f \in ts\ \}$$

Here, `IMAGE ($, T) ts` applies the pair-constructor `($, T)` to each member f of ts, turning f into the T-signed formula (T, f). Similarly, `IMAGE ($, F) fs` turns every member f of fs into the F-signed formula (F, f).

The function mk_seq accepts a set of pairs where, in each pair (a, b), the a is of type bool (a sign) and b is of type 'f (an unsigned formula in the case that 'f is 'a formula). It produces a result which is a pair (L, R) of sets where L (the antecedent) and R (the succedent) are both sets of type 'f set. That is, (L, R) is the sequent $L \vdash R$. For some given set S of signed formulae, Fitting would write these as $L = (S^F)^\circ$ and $R = (S^T)^\circ$ respectively.

The function dest_seq does the reverse: it accepts a pair (L, R) of sets, each of type 'f set, and produces a result which is a set of pairs where, in each pair (a, b), the a is of type bool and the b is of type 'f. When 'f is 'a formula, this gives us a set of signed formulae of type bool # 'a formula.

We can convert between the two formalisms via the following lemma.

Lemma 1. *Let* sf *be a set of signed formulae and let* sq *be a sequent (a pair of sets of unsigned formulae). Then* (dest_seq sq = sf) *iff* (mk_seq sf = sq)*.*

```
(dest_seq sq = sf) <=> (mk_seq sf = sq)
```

3.4 Rule-skeletons, contexts and rules

A dual-tableau rule consists of a single premise, which is a set of signed formulae, and multiple conclusions, each of which is a set of signed formulae. The premise typically has a single principal formula and each conclusion has possibly multiple side-formulae. There is usually also a context S which remains unchanged from premise to conclusions. For example, consider the rule for $F\ X \vee Y$ shown below on the left. It consists of a rule-skeleton as shown below on the right, which is then decorated uniformly by the context S to give the actual rule shown on the left. Its principal formula is $F\ X \vee Y$ and its side-formula sets are $\{F\ X, F\ X \vee Y\}$ and $\{F\ Y, F\ X \vee Y\}$:

$$\frac{S; F\ X \vee Y}{S; F\ X \vee Y; F\ X \qquad S; F\ X \vee Y; F\ Y} \qquad \frac{F\ X \vee Y}{F\ X \vee Y; F\ X \qquad F\ X \vee Y; F\ Y}$$

This describes Fitting's first six rules [Fit17, Figure 1.6].

Definition 5 (rule skeleton)**.** *A rule-skeleton of type* 'a rule_sk *is a pair* (psk, cssk) *consisting of a single item* psk *of type* 'a *and a set* cssk *of sets of items of type* 'a *using the type abbreviation below:*

```
val _ = Parse.type_abbrev ("rule_sk", ``: ('a # 'a set set)``) ;
```

Ensuring that 'a is always a signed formula type 'b sf then restricts rule skeletons to be over a signed formula and a set of sets of signed formulae.

For example, consider the rule skeleton for $F\ X \vee Y$ shown above at right. The intuition is that the first component psk will encode the single formula $F\ X \vee Y$ in the premise of the rule-skeleton while the second component cssk of the pair will encode the set $\{\{F\ X \vee Y, F\ X\}, \{F\ X \vee Y, F\ Y\}\}$ of sets $\{F\ X \vee Y, F\ X\}$ and $\{F\ X \vee Y, F\ Y\}$ of signed formulae.

Definition 6. *A rule of type* 'a rule *is a pair* (p, cs) *consisting of a (premise) set* p *of type* 'a *and a (conclusions) set* cs *of terms of type* 'a*.*

```
val _ = Parse.type_abbrev ("rule", ``: ('a # 'a set)``) ;
```

The intuition is that the first component p will encode the formula set $S \cup \{F\ X \vee Y\}$ as the premise of the rule while the second component cs of the pair will encode the set $\{\ S \cup \{F\ X \vee Y, F\ X\}\ ,\ S \cup \{F\ X \vee Y, F\ Y\}\ \}$ of sets $S \cup \{F\ X \vee Y, F\ X\}$ and $S \cup \{F\ X \vee Y, F\ Y\}$ of signed formulae.

Thus `rule_sk` (rule-skeleton) models the operation that has a premise of type `'a` (here, a signed formula) and a set of conclusions where each conclusion is a set of items of type `'a` (here, signed formula), while `rule` (a rule, with context, as applied) models an operation that has a premise of type `'a` (here, a set of signed formulae) and a set of conclusions, each of which is an item of type `'a` (here, a set of signed formulae).

Definition 7. *For all s and st,* ***is_tab_rule*** (s, st) *returns the set of pairs of the form*

$$(\{s\} \cup U,\ \{U \cup t \mid t \in st\})$$

```
is_tab_rule : 'sf rule_sk -> 'sf set rule set
!s st U. is_tab_rule (s, st) ({s} UNION U, IMAGE ($UNION U) st)
```

Intuitively, the relation `is_tab_rule (s, st)` applies a context `U` to the premise `s` and to each branch of the conclusion `st` of such a rule-skeleton. Here we choose to write the type variable as `'sf`, to suggest that, while `is_tab_rule` can be used for any type, we will use it for the signed formula type. Also `A UNION B` encodes $A \cup B$. The `!` is the universal quantifier $\forall$, while `?` is the existential quantifier $\exists$. The function `IMAGE x y` returns the result of applying the function `x` to every member of the set `y` and `($UNION U)` is the function that forms the union of its argument with the set U: thus `IMAGE ($UNION U) st` encodes $\{U \cup t \mid t \in st\}$.

The type given for `is_tab_rule` uses the type abbreviations given above for `rule_sk` and `rule` which means that the type `'sf rule_sk` is an abbreviation for `'sf # 'sf set set`, and `'sf set rule` is an abbreviation for `'sf set # 'sf set set`.

So where a rule-skeleton, as in [Fit17, Fig. 1.3], has, for its premise, a signed formula and, for its conclusion, a set of sets of signed formulae (so of type `'sf rule_sk`), adding context gives a rule as in [Fit17, Fig. 1.6], which has, for its premise, a set of signed formulae and, for its conclusion, a set of sets of signed formulae (so of type `'sf set rule`).

The definition of `is_tab_rule` is an inductive definition, which means that `is_tab_rule` is defined to be the predicate which is satisfied (only) by pairs that can be inferred to satisfy it using the definition clause given.

For rules such as the last two of [Fit17, Fig. 1.6], where only F-signed context items are allowed in the result, we have similar relations:

Definition 8. *For all s, st and U,* ***is_ag_tab_rule*** (s, st) *returns the pair*

$$(\{s\} \cup U,\ \{t \cup U_F \mid t \in st\}) \text{ where } U_F = \{(F, g) \mid (F, g) \in U\}$$

```
is_ag_tab_rule : 'a sf rule_sk -> 'a sf set rule -> bool ;
!s st U. is_ag_tab_rule (s, st)
         ({s} UNION U, IMAGE ($UNION (U INTER ($~ o FST))) st)
```

Here, the construct `~` is boolean negation, meaning that it returns true iff its argument is of type `bool` and is `F`. The construct `o` is relational composition and

FST is the function that returns the first component of a pair. So the construct `($~ o FST)` is effectively the set of all F-signed formulae. The construct `INTER` is set intersection so `(U INTER ($~ o FST))` is the set of F-signed formulae from `U`. Thus `is_ag_tab_rule` allows any context in the premise, but only F-signed context in the conclusion (as in the last two rules of [Fit17, Figure 1.6]),

Definition 9. *For all s, st and U, `is_aa_tab_rule` (s, st) returns the pair*

$$(\{s\} \cup U_F,\ \{t \cup U_F \mid t \in st\}) \textit{ where } U_F = \{(F, g) \mid (F, g) \in U\}$$

```
is_aa_tab_rule : 'a sf rule_sk -> 'a sf set rule -> bool ;
!s st U. is_aa_tab_rule (s, st)
          ( {s} UNION (U INTER ($~ o FST)) ,
            IMAGE ($UNION (U INTER ($~ o FST))) st )
```

Thus `is_aa_tab_rule` allows F-signed contexts only, in the premise and conclusion (this is useful for a lemma we need).

Each rule is defined in its skeletal form (without the context). For example, the rule-skeleton for $F\ X \vee Y$ shown above is encoded as below resulting in the type shown:

```
!X Y. or_left (
  (F, Or X Y),
  { {(F, Or X Y); (F, X)} ; {(F, Or X Y); (F, Y)} }
  )
or_left : 'a sf rule_sk set
or_left : ((bool # 'a formula) # ((bool # 'a formula) set set)) set
```

We then collect these rule skeletons into two sets as below.

Definition 10 (`gen_idt_rule`, `ant_idt_rule`)**.** *The following sets of rules are defined in skeleton form:*

`gen_idt_rule`: *skeleton form of the six rules which allow arbitrary contexts;*

`ant_idt_rule`: *the skeletons of the `imp_right` and `not_right` rules which have only a F-signed context in the result (see [Fit17, Figure 1.6]).*

We then define all the rules of the system by taking these skeletons and allowing contexts appropriately as below.

Definition 11 (`idt_tab_rule`)**.** *The set `idt_tab_rule` is inductively defined via the two clauses below:*

`gen_idt_rule gr`: *For all `gr`, if `gr` is the skeleton of a rule from the first 6 rules of Fig 6, and `rl` is obtained from `gr` by adding an arbitrary context then `rl` is a rule of the dual-tableau calculus.*

`ant_idt_rule`: *For all `gr`, if `gr` is the skeleton of a rule from the last two rules of Fig 6, and `rl` is obtained from `gr` by adding an arbitrary context to the premise but adding only the `F`-signed part of this context to the conclusions then `rl` is a rule of the dual-tableau calculus.*

```
idt_tab_rule : 'a sf set rule set

(!gr rl. gen_idt_rule gr
       /\ is_tab_rule gr rl ==> idt_tab_rule rl)
/\
(!gr rl. ant_idt_rule gr
       /\ is_ag_tab_rule gr rl ==> idt_tab_rule rl);
```

Here, `gr` will be a pair consisting of the skeleton premise and the skeleton conclusions while `rl` will be those pairs, each extended by some appropriate context. Again, the above is an inductive definition of the rules `idt_tab_rule` for intuitionistic dual-tableaux so these are the only ways to obtain a legal rule.

Notice that we do not define a closed dual-tableau, which is dealt with in the next subsection.

3.5 Branches, dual-tableaux and their fringes

Each branch of a dual-tableau ends in a leaf which is a set of signed formulae. So the set of all leaves of a dual-tableau, which we will call its "fringe", is a set of sets of signed formulae. When we apply a rule to one of these leaves, the effect on the fringe is to replace that single leaf by the set of leaves which is the result of the rule.

Definition 12 (`extend_fringe`)**.** *For all s, sfr and rule sets rs, if rs contains a rule which takes s to the set sfr, and we apply it to a dual-tableau with a fringe consisting of the leaf s plus the leaf items rf of the other branches, in addition to s, then the result is the new leaves sfr arising from s plus the unchanged leaves rf of the other branches.*

```
! rs s sfr. rs (s, sfr)
          ==> extend_fringe rs ({s} UNION rf, sfr UNION rf) ;
extend_fringe : 'sfs rule set -> ('sfs set # 'sfs set) set ;
```

So for a rule set `rs`, the function `extend-fringe rs` gives the set of resulting transformations of the fringe of a dual-tableau obtained by applying one of the rules (s, sfr). Here, we have written the type variable as `'sfs` to suggest that we will use `extend-fringe` where `'sfs` is the type of sets of signed formulae. (We will also use the term variable `sfs` to indicate a set of signed formulae). Moreover, we shall instantiate `rs` as `idt_tab_rule` giving the type:

```
extend_fringe idt_tab_rule : ('a sf set set # 'a sf set set) set ;
```

At this point we note that in `HOL4`, sets and predicates are identified and so $x \in P$, (*i.e.* `x IN P`), means exactly $P\ x$ (*i.e.* `P x`). Consequently, Definition 12 of `extend_fringe` might be more clearly written:

```
(s, sfr) IN rs
           ==> ({s} UNION rf, sfr UNION rf) IN extend_fringe rs
```

This is an inductive definition which means that `extend_fringe rs` is defined to be the set of those fringe-transformations which can be inferred to be in that set by application of this definition.

The intuition is that we do not keep track of the internal nodes of a dual-tableau: we keep track of its root (a set of signed-formulae) and its fringe (a set of sets of signed-formulae).

3.6 Closed dual-tableaux and a statement of soundness

By definition, a branch tip, or leaf, is then just a member of the fringe of a dual-tableau, that is, a leaf is a set of signed formulae.

Definition 13 (closed branch)**.** *A branch tip, i.e. a leaf,* `sfs` *is closed if it contains some formula* `f` *signed* `F` *and* `T`*:*

```
br_closed : 'a sf set -> bool
br_closed sfs = ?f. (T, f) IN sfs /\ (F, f) IN sfs
```

A dual-tableau (fringe) `sfss` *is closed if every branch* `sfs` *in it is closed:*

```
dt_closed : 'a sf set set -> bool
dt_closed sfss = !sfs. sfs IN sfss ==> br_closed sfs
```

A leaf `sfs` *is atomically closed if it contains some atomic formula* `Atom p` *signed* `F` *and* `T`*:*

```
at_closed : 'a sf set -> bool
at_closed sfs = ?p. (T, Atom p) IN sfs /\ (F, Atom p) IN sfs
```

Note that we work with closure defined using T- and F-signed occurrences of an arbitrary formula f rather than an atomic formula `Atom` p. Later on (see Lemma 22) we show that everything still goes through if we demand that f is atomic.

Now, the action of repeatedly applying dual-tableau rules from some set of rules can be expressed as the reflexive transitive closure of application of any rule from that set. In `HOL4`,, a reflexive transitive closure function `RTC` is provided: it takes and returns relations of the type `'a -> 'a -> bool`. For a relation R of this type, aRb is expressed in `HOL4` as $R\ a\ b$.

Thus our soundness theorem will be of the form below:

> If (R, pv) is an intuitionistic Kripke model and if the dual-tableau fringe *bot* is obtained from repeated applications of the set `idt_tab_rule` of rules to the initial fringe $\{\{(T, f)\}\}$, <u>and</u> *bot* is closed then the formula f is true in every world w of (R, pv):
>
> ```
> Kripke_model R pv ==>
>    RTC (CURRY (extend_fringe idt_tab_rule)) {{(T,f)}} bot ==>
>    dt_closed bot ==> forces R pv w f
> ```

Here, `CURRY` : $(\alpha \ \times \ \beta \rightarrow \mathit{bool}) \rightarrow \alpha \rightarrow \beta \rightarrow \mathit{bool}$ takes a relation in the form of a predicate of type $(\alpha \ \times \ \beta \rightarrow \mathit{bool})$ on pairs (α, β), and returns a relation in the form of the same predicate with the type $(\alpha \ \rightarrow \ \beta \rightarrow \mathit{bool})$ on two curried arguments α and β: which is the form required by `RTC`. Also, note that the construct `A ==> B ==> C` in `HOL4` is logically equivalent to the construct `A /\ B ==> C`, which is why the English prose uses "<u>and</u>" rather than "implies".

Notice that we defined a (portion of a) dual-tableau using reflexive transitive closure of the relation which takes one fringe to the next, so the initial fringe is a singleton set $\{\{(T, f)\}\}$ containing the initial leaf $\{(T, f)\}$.

Note: some of the rules copy their principal formulae into all of their conclusions. So why do we not have to worry about termination in the soundness proof? Because, by definition, the reflexive-transitive closure is obtained by a finite number of applications, thus each dual-tableau is finite by definition.

To complete this theorem, we now have to formalise the Kripke semantics of intuitionistic logic, thereby formalising the notions of `Kripke_model R pv` and `forces R pv w f`.

4 Formalised intuitionistic Kripke models

The Kripke semantics of intuitionistic logic are based upon classical logic, so we can encode these semantics directly into the classical higher-order logic of `HOL4`.

Using `R` to encode the underlying binary relation, using `v` and `w` for worlds, and using `pv` for the (classical) propositional valuation of an atom `a` to one of true or false at a world, we define a persistent valuation function directly:

Definition 14. *A binary relation R over some given set of worlds of type* `'w` *is a function that maps a pair w and v of worlds of type* `'w` *to* `T` *or* `F` *depending on whether the pair* (w, v) *is or is not in the relation, ie, whether* wRv *or not.*

A propositional valuation pv maps a world w of type `'w` *and an atom of type* `'a` *to* `T` *or else* `F` *depending on whether the atom a is true or false at world w.*

```
R  : 'w -> 'w -> bool
pv : 'w -> 'a -> bool
```

Definition 15 (`persistent R pv`)**.** *The classical valuation* `pv` *is persistent over a binary relation* `R` *over some set of worlds if for all worlds* `v` *and* `w`*, and all atoms* `a`*, if* `w` *is an* `R`*-successor of* `v` *then if* `a` *is true at* `v` *then* `a` *is true at* `w`*:*

```
persistent R pv = !v w a. R v w ==> pv v a ==> pv w a
```

Using the predicates `transitive` and `reflexive` which are pre-defined in `HOL4`, we define `R, pv` to be a Kripke model as follows.

Definition 16 (Kripke_model)**.** `R` *and* `pv` *is a Kripke model iff the binary relation* `R` *is reflexive and transitive, and the valuation* `pv` *is persistent over* `R`*:*

```
Kripke_model R pv
         = transitive R /\ reflexive R /\ persistent R pv
```

Note, currently there is no condition that the set of worlds is non-empty as is usual in Kripke semantics. For the moment, we do not need it. As we shall see, it will become essential later in the completeness proof. Also, note that intuitionistic Kripke frames are often defined to be reflexive, transitive and anti-symmetric: $\forall x, y. R\ x\ y\ \&\ R\ y\ x \implies x = y$. The "persistence" of the binary relation R ensures that the two definitions give rise to the same notion of validity. But again, we do not require this extra condition.

Definition 17 (`forces R pv w f`)**.** *The usual forcing relation* `forces R pv w f` *that holds between a model* `R, pv`*, a world* `w` *and a formula* `f` *is then as defined below:*

```
   (forces R pv w (Atom a)  = pv w a)
/\ (forces R pv w (And p q) = forces R pv w p /\ forces R pv w q)
/\ (forces R pv w (Or p q)  = forces R pv w p \/ forces R pv w q)
```

```
/\ (forces R pv w (Not p)   = !v. R w v ==> ~ forces R pv v p)
/\ (forces R pv w (Imp p q) =
              !v. R w v ==> forces R pv v p ==> forces R pv v q)

forces : ('w -> 'w -> bool) -> ('w -> 'a -> bool)
                            -> 'w -> 'a formula -> bool ;
```

We say a world v is a *future* world of world w if $R\ w\ v$.

Lemma 2 (FORCES_PERSISTENT)**.** *If the binary relation* `R` *is transitive and the valuation* `pv` *of atomic formulae is persistent over* `R` *then so is the forcing predicate* `forces R pv`*:*

```
transitive R ==> persistent R pv ==> persistent R (forces R pv)
```

In the above, the two uses of `persistent` have different types, the first is about a valuation of atoms while the second is about the forcing predicate (which is a derived valuation of formulae).

We obtain an equivalent version of FORCES_PERSISTENT:

Lemma 3 (FORCES_IF_ALL)**.** *If* `R, pv` *is a Kripke model then a world* `w` *in the model forces a formula* `f` *if and only if every future world* `v` *forces* `f`*.*

```
Kripke_model R pv ==>
      ((!v. R w v ==> forces R pv v f) = forces R pv w f)
```

5 Attributed formulae and soundness

The proof of soundness involves attributing an intuitionistic formula to each signed-formula set in a fringe of the dual-tableau, and proving that the rules preserve intuitionistic validity of these attributed formulae upwards: that is, for each rule, if each intuitionistic formula attributed to a conclusion of that rule is intuitionistically valid then so is the intuitionistic formula attributed to the premise of that rule.

We first tried to encode this notion of a valuation for the attributed formula directly but got stuck when, contrary to our expectations, we found that Lemma 7 does not hold for these valuations. We therefore reworked all the definitions as shown next.

Given a set *sfs* of signed formulae, the intuitionistic formula attributed to *sfs* is $\bigwedge Fs \supset \bigvee Ts$ where *Fs* and *Ts* are each the set of unsigned formulae that are F-signed and T-signed in *sfs*, respectively. Here, the empty disjunction is read as contradiction $\perp$ and the implication $p \supset \perp$ is intuitionistically equivalent to the negation of p. According to Definition 17, the intuitionistic semantics of $p \supset q$ (`Imp p q`) at a world w involves evaluating the classical logic implication `forces R pv v p ==> forces R pv v q` over all R-successors `v`, so we first encode this "classical" notion and put Fs for p and Ts for q.

Definition 18. *The predicate* `sfs_val_aux R pv v sfs` *is true iff: if every formula f signed F in the set sfs of signed-formulae is forced at v then some formula t signed T in the set sfs of signed-formulae is also forced at v.*

```
sfs_val_aux R pv v sfs =
  let (Fs, Ts) = mk_seq sfs in
  (!f. f IN Fs ==> forces R pv v f)
                                ==> (?t. t IN Ts /\ forces R pv v t)
```

Here, we first convert the set `sfs` of signed formulae into a sequent $Fs \vdash Ts$ using our previously defined function `mk_seq`. Then we encode the classical logic formula $\forall f \in Fs.$ `forces` $R\ pv\ v\ f \Rightarrow \exists t \in Ts.$ `forces` $R\ pv\ v\ t$ rather than the intuitionistic formula $\bigwedge Fs \supset \bigvee Ts$ attributed to *sfs*. To obtain the valuation of the attributed formulae, we have to evaluate this auxiliary classical formula over all future worlds.

Definition 19. *The predicate* `sfs_val R pv w sfs` *holds iff every future world v of w satisfies* `sfs_val_aux R pv v sfs`*:*

```
sfs_val R pv w sfs = !v. R w v ==> sfs_val_aux R pv v sfs
```

Definition 20. *The valuation of the conclusions fss of a branching rule is the conjunction of the valuations of the signed formula sets sfs in each conclusion. The valuation of a dual-tableau fringe fss is the conjunction of the valuations of each constituent leaf sfs. Both notions can be captured by instantiating the definition below appropriately:*

```
tab_val R pv w fss = (!sfs. sfs IN fss ==> sfs_val R pv w sfs)
```

Again, it is also useful to define a corresponding auxiliary function giving the "classical" valuation of the whole dual-tableau fringe, or of the conclusions of a rule, at a particular world v:

Definition 21. *The predicate* `tab_val_aux R pv v fss` *holds of a fringe* `fss` *iff every leaf* `sfs` *in the fringe satisfies* `sfs_val_aux R pv v sfs`*:*

```
tab_val_aux R pv v fss
     = (!sfs. sfs IN fss ==> sfs_val_aux R pv v sfs)
```

The open loop below captures that `tab_val_aux` is defined in terms of `sfs_val_aux` which is used to define `sfs_val` which is used to define `tab_val`:

```
sfs_val_aux --- sfs_val
     |             |
tab_val_aux      tab_val
```

The next lemma "closes the loop" by expressing `tab_val` in terms of `tab_val_aux`, stating that `tab_val` does indeed evaluate `tab_val_aux` over all future worlds.

Lemma 4 (`tab_val_alt`)**.** *The predicate* `tab_val R pv w fss` *holds iff the auxiliary predicate* `tab_val_aux R pv v fss` *holds at every future world v of w:*

```
tab_val R pv w fss = !v. R w v ==> tab_val_aux R pv v fss
```

We want to prove soundness in terms of closed dual-tableaux, so we have

Lemma 5 (`idt_br_sound`,`idt_dt_sound`)**.** *If a dual-tableau branch is closed, then the valuation (using* `sfs_val`*) of the leaf of that branch is true, and if a dual-tableau is closed, then the valuation (using* `tab_val`*) of the fringe is true.*

```
idt_br_sound : br_closed br ==> sfs_val R pv w br
idt_dt_sound : dt_closed tab ==> tab_val R pv w tab
```

The intuition of the above lemma is, of course, that the leaf of each closed branch contains at least one formula f that appears in both Fs and Ts, and hence is the witness for the right-hand side of $\bigwedge Fs \supset \bigvee Ts$.

For the first six rules of [Fit17, Figure 1.6] (without the context S) the preservation of validity is in fact an equivalence where we use b as a placeholder for a sign:

Lemma 6 (idt_rules_aux_eqv). *For the skeletons of the first six rules of [Fit17, Figure 1.6] (without the context S), the auxiliary valuation of the signed formula (b, f) from the rule premise equals the auxiliary valuation of the set sfss of signed formula sets from the conclusions of the rule, as long as R is reflexive:*

```
reflexive R ==> gen_idt_rule ((b,f),sfss)
    ==> (tab_val_aux R pv w sfss = sfs_val_aux R pv w {(b,f)})
```

Here, we have deliberately used `(b, f)` rather than the equivalent `sf` to highlight the following: why do we suddenly need reflexivity? Because for the $\supset$-F rule (ignoring the S), where $b = F$ and $f = (X \supset Y)$, the `tab_val_aux` of the conclusions $F(X \supset Y), FY$ and $F(X \supset Y), TX$ is the conjunction of the respective semantic clauses $w \Vdash X \supset Y \Rightarrow w \nVdash Y$ and $w \Vdash X \supset Y \Rightarrow w \Vdash X$ while the `sfs_val_aux` of the premise $F(X \supset Y)$ is $w \nVdash X \supset Y$. For the former to imply the latter we require $w \nVdash Y$ and $w \Vdash X$ to imply $w \nVdash X \supset Y$. But $w \nVdash X \supset Y$ is $\exists v.\ wRv\ \&\ v \Vdash X\ \&\ v \nVdash Y$, which holds if we choose $v = w$ by reflexivity of R.

Adding the context preserves this property:

Lemma 7 (is_tab_rule_pres_eqv). *If a dual-tableau rule* `(sfs, sfss)` *preserves auxiliary valuations, then the extension* `(esfs, esfss)` *of that rule by a context also preserves them.*

```
is_tab_rule (sf, sfss) (esf, esfss) ==>
  (tab_val_aux R pv w sfss = sfs_val_aux R pv w {sf}) ==>
    (tab_val_aux R pv w esfss = sfs_val_aux R pv w esf)
```

Here, notice that we first need to turn the single signed-formula sf into a set $\{sf\}$ of signed-formulae while esf is a set of signed-formulae since it is an extension of sf by adding context.

We tried to prove Lemma 7 for the actual ("non-auxiliary") valuations, but it doesn't hold.[1] This means that the proof of Lemma 13, so far as it concerns these rules, depends on first applying Lemma 7 to Lemma 6 and only then quantifying over future worlds.

However Lemma 6 clearly extends to the actual valuations, and we get a similar equivalence for the last two rules of [Fit17, Figure 1.6].

[1] The predicates `tab_val_aux R pv` x `sfss` and `sfs_val_aux R pv` y `{sf}` may be false only at worlds $x =$ `u` and $y =$ `v` respectively, where `u` and `v` are different future worlds of `w`, which make `tab_val_aux R pv w sfss` and `sfs_val_aux R pv w {sf}` equal (both false); however adding context may change the valuation to make it true at world `u` but not `v`, or vice versa, which would make `tab_val_aux R pv w esfss` and `sfs_val_aux R pv w esf` unequal. This doesn't suggest a flaw in Fitting's proof, rather that the level of detail he gives doesn't indicate precisely the sequence of lemmata to be used.

Lemma 8 (ant_rules_eqv). *For the skeletons of the last two rules of [Fit17, Figure 1.6] (without the context S or S_F), the valuation of the premise signed formula equals the valuation of the set of signed formula sets in the conclusions, as long as the relation is reflexive and transitive:*

```
transitive R ==> reflexive R ==>
   ant_idt_rule ((b, f), sfss) ==>
       (tab_val R pv w sfss = sfs_val R pv w {(b,f)})
```

Note that we now need both reflexivity and transitivity.

We can characterise the effect of adding antecedent context, that is, adding F-signed formulae to the context:

Lemma 9 (ant_ctxt_eqv). *For a set U of signed formulae, if we add the set U_F of all F-signed formulae from U to a signed formula set sfs then the valuation of the augmented set $U_F \cup sfs$ is given by: $w \Vdash U_F \cup sfs$ iff forall v such that wRv, if $v \Vdash (F, f)$ for all $(F, f) \in U_F$ then $v \Vdash sfs$.*

```
Kripke_model R pv ==>
 (sfs_val R pv w ((U INTER $~ o FST) UNION sfs) =
  !v. R w v ==>
  (!f. (F, f) IN U ==> forces R pv v f) ==> sfs_val R pv v sfs)
```

Here, the right-hand side of the equality intuitively captures the valuation of the attributed formula $U_F \supset (S_F \supset S_T)$ where `mk_seq sfs =` (S_F, S_T) since the outermost quantification over future worlds v captures the outermost occurrence of $\supset$, and the use of `sfs_val` captures the inner occurrence of $\supset$. The left-hand side of the equality intuitively captures the valuation of the attributed formula $(U_F \wedge S_F) \supset S_T$ since the inner $\wedge$ is handled by the `UNION` operation and the outer $\supset$ is handled by the quantification over future worlds inside `sfs_val`. Thus it relies on the intuitionistic logic theorem $((A \wedge B) \to C) \leftrightarrow (A \to B \to C)$. A similar characterisation of the effect of adding succedent context is not available because it is not the case that $S_F \supset (S_T \vee U_T)$ is expressible (intuitionistically) as $(S_F \supset S_T)$ *op* U_T for any operator *op*.

So we get the following result, that if a dual-tableau rule preserves the valuation (at all future worlds), then the extension of that rule by adding antecedent context preserves the valuation (at the present world).

Lemma 10 (is_aa_tab_rule_pres_eqv). *If, at all future worlds, the valuation of the conclusions sfss of a rule equals the valuation of the premise sf of the rule, then, when the rule is extended with antecedent context, the valuation of the extended conclusions esfss equals the valuation of the extended premise esf.*

```
Kripke_model R pv
  ==> is_aa_tab_rule (sf, sfss) (esf, esfss)
  ==> (!v. R w v ==> (tab_val R pv v sfss = sfs_val R pv v {sf}))
  ==> (tab_val R pv w esfss = sfs_val R pv w esf)
```

Why is it $\{sf\}$ but not $\{esf\}$? Because sf is a single signed formula and `sfs_val` requires a set of signed-formulae, while esf is a set of signed-formulae since it is $\{sf\}$ extended by adding context.

To get from this to the case where the premise can have an arbitrary context, we just need weakening as shown next.

Lemma 11 (sfs_val_wk_sub). *If a signed formula set A has valuation true, then so does any signed formula superset C of A.*

```
A SUBSET C ==> sfs_val R pv v A ==> sfs_val R pv v C
```

Combining all these results we get the "upward" preservation of valuations from the conclusions of a rule to its premise that we seek.

Lemma 12 (idt_pres). *If we apply a rule to a dual-tableau branch leaf sfs, and the resulting conclusions sfss have valuation true, then the branch leaf sfs has valuation true.*

```
Kripke_model R pv
    ==> idt_tab_rule (sfs, sfss)
    ==> tab_val R pv w sfss
    ==> sfs_val R pv w sfs
```

Now we get the corresponding result for the application of a rule to the fringe of a dual-tableau, rather than a single leaf (set of signed formulae).

Lemma 13 (idt_pres_frg). *If we apply a rule to a dual-tableau fringe* `prev`*, and the resulting fringe* `next` *has valuation true, then so does* `prev`*:*

```
Kripke_model R pv  ==>
  extend_fringe idt_tab_rule (prev, next) ==>
     tab_val R pv w next  ==> tab_val R pv w prev
```

A similar result also holds for the reflexive transitive closure of the set of rules, not just a single rule.

Lemma 14 (idt_rtc_pres_frg). *If we apply a sequence of rules to an initial dual-tableau fringe top, and the resulting fringe bot has valuation true, then so does the starting fringe top:*

```
Kripke_model R pv ==>
   !bot. RTC (CURRY (extend_fringe idt_tab_rule)) top bot ==>
       tab_val R pv w bot ==> tab_val R pv w top
```

For a dual-tableau proof of formula *f*, the starting point (the initial fringe) is $\{\{(T, f)\}\}$ and we have the following lemma.

Lemma 15 (tab_val_single). *The dual-tableau valuation for the fringe* $\{\ \{\ (T, f)\ \}\ \}$ *at a world* w *of a Kripke model* R, pv *is true iff the world* w *forces* f*.*

```
Kripke_model R pv ==>
           (tab_val R pv w {{(T,f)}} = forces R pv w f)
```

Finally, the soundness result, using Lemmas 14, 15 and 5.

Theorem 1 (idt_sound). *If a dual-tableau for the the signed formula* (T, f) *is closed then any model R, pv forces f at any world w.*

```
Kripke_model R pv ==>
   RTC (CURRY (extend_fringe idt_tab_rule)) {{(T,f)}} bot
   ==> dt_closed bot
   ==> forces R pv w f
```

Here, the dual-tableau has an initial fringe $\{\{(T, f)\}\}$ and repeatedly applying the dual-tableau rules to this fringe converts it to the closed fringe *bot*. Why do we not have explicit universal quantifiers over R, pv and w? Because every "free" variable in a statement is automatically considered by `HOL4` to be universally quantified.

6 Formalising completeness

We now describe how we formalised Fitting's completeness proof [Fit17, Section 1.3.3] for intuitionistic dual tableaux.

6.1 I-tautologous sets of signed formulae

We give two ways to formalise I-tautologous sets, one following Fitting and another using inductive definitions.

6.1.1 I-tautologous sets

We first define an I-tautologous set of signed formulae similarly to [Fit17, Definition 7], but without the requirements that: (i) an I-tautologous set S must have a finite I-tautologous subset; and (ii) that the closure of each branch is atomic; and (iii) that dual-tableaux satisfy the single-use restriction, whereby only active signed formulae are considered for a rule application (see [Fit17, Definitions 1,4]). We also define an I-tautologous set of sets of signed formulae (a set of dual-tableau leaves, *i.e.* a dual-tableau fringe):

Definition 22 (Itautss,Itauts)**.** *A set `top` of sets of signed formulae is I-tautologous w.r.t. a set `rs` of rules (i.e. `Itautss rs top` holds) if starting with `top` and repeatedly applying rules from `rs` gives a fringe `bot` which is closed. A set `s` of signed formulae is I-tautologous w.r.t. a set `rs` of rules if `Itautss rs` {`s`} holds.*

```
Itautss : 'a sf set rule set -> 'a sf set set -> bool ;
Itautss rs top =
  ?bot. RTC (CURRY (extend_fringe rs)) top bot /\ dt_closed bot

Itauts  : 'a sf set rule set -> 'a sf set -> bool ;
Itauts rs s = Itautss rs {s}
```

We proved

Lemma 16 (ITAUTSS_ALL)**.** *A finite set `sfss` of sets of signed formulae is I-tautologous if and only if each of its member sets `sfs` is I-tautologous.*

```
FINITE sfss ==>
     Itautss rs sfss <=> (!sfs. sfs IN sfss ==> Itauts rs sfs)
```

Lemma 17 (ITAUT_EX_RULE)**.** *Assuming a set `rs` of rules is finitely branching, a set `top` of signed formulae is I-tautologous w.r.t. `rs` iff it is closed, or there is a dual-tableau rule `(top, rb)` which can be applied to it and every resulting branch `br` in the conclusion `rb` is I-tautologous.*

```
IMAGE SND rs SUBSET FINITE ==>
  (Itauts rs top = dt_closed {top}
   \/ ?rb. (top, rb) IN rs /\ !br. br IN rb ==> Itauts rs br)
```

Here, `rs` is a set of rules and `IMAGE SND rs` is the set of second components (results) of those rules. Thus `IMAGE SND rs` is the set $\{C_1, C_2, \cdots\}$ where $C_i = \{c_1^i, c_2^i, \cdots\}$ is the set of conclusions of some rule from `rs`, where each conclusion c_j^i is a set of signed formulae. The construct `FINITE` is the set of all finite sets so `X SUBSET FINITE` encodes $\forall x \in X.\ x \subset$ `FINITE`. Thus `IMAGE SND rs SUBSET FINITE` says that each C_i is finite, which captures that each rule is finitely branching.

6.1.2 An inductive definition of I-tautologous sets

Lemma 17 seems obvious, but was difficult to prove in `HOL4`, so we tried to reformulate the definition to make the mechanics of the `HOL4` proofs easier. We therefore defined an I-tautologous set of signed formulae as an inductively defined set, using the fact stated in Lemma 17.

Definition 23 (`Itauti`). *For every rule set `rs`, a set `top` of signed formulae satisfies `Itauti rs` iff*

(i) `top` is itself closed, or

(ii) some rule `(top, rb)` in `rs` is applicable to `top` to obtain the conclusion `rb` and every resulting branch `br` in `rb` is I-tautologous w.r.t. `rs`

and `Itauti rs` is the unique minimal predicate (set) such that (i) and (ii) hold.

```
(!top. br_closed top ==> Itauti rs top) /\
(!top. (?rb. (top,rb) IN rs /\ !br. br IN rb ==> Itauti rs br)
                        ==> Itauti rs top)
```

First, note that the linguistic "or" between clauses (i) and (ii) turns into a logical "and" (/\) because the English clauses capture the equivalent definition:

```
!top.( (br_closed top) \/
       (?rb. (top,rb) IN rs /\ !br. br IN rb ==> Itauti rs br)
     ) ==> Itauti rs top
```

However we used the more common and, for proofs, useful, style of definition, with multiple clauses. Second, by using `HOL4`'s inductively defined sets, the assertion contained in the definition (that there is a unique minimal such predicate) is proved automatically by `HOL4`, as expressed in Lemma 18.

Lemma 18 (`Itauti_ind`). *For all rule sets `rs` and all predicates `Itauti'` on signed formula sets, if*

(i) every closed signed formula set `top` satisfies `Itauti'`, and

(ii) whenever `(top, rb)` is a rule in `rs`, and every signed formula set `br` in the rule conclusion `rb` satisfies `Itauti'`, then `top` satisfies `Itauti'`

then every signed formula set `a0` satisfying `Itauti rs` satisfies `Itauti'`.

```
!rs Itauti'.
 (!top. br_closed top ==> Itauti' top) /\
 (!top. (?rb. (top,rb) IN rs /\ !br. br IN rb ==> Itauti' br)
        ==> Itauti' top)
 ==> !a0. Itauti rs a0 ==> Itauti' a0
```

Intuitively, the lemma states that any set (predicate) `Itauti'` which is closed under the clauses of Definition 23 for `Itauti rs` is a superset of `Itauti rs`: *i.e.* `Itauti rs` is the smallest set satisfying those clauses.

6.1.3 Relating the two notions of I-tautologous sets

We then proved the equivalence of Definition 23 for `Itauti` and Definition 22 for `Itauts`, under the assumption that rules are finitely branching: this assumption is required since the definition `Itauti` allows the case of an infinitely branching dual-tableau even without any path of infinite depth down a branch.

Lemma 19 (ITAUTS_EQ_I). *For every rule set* `rs`*, if the rules in* `rs` *are finitely branching then the properties* `Itauti` *and* `Itauts` *are equivalent.*

```
IMAGE SND rs SUBSET FINITE ==> (Itauts rs = Itauti rs)
```

Note that the equivalence does not hold for (even a finite set of) infinitely branching rules because an infinitely branching rule can give an infinite dual-tableau (which may be of unbounded depth), in which each path down a branch is finite. If such a dual-tableau is closed then it meets the definition of `Itauti`, but not the definition of `Itauts` since it does not close in a finite number of rule applications; to put it another way, definition `Itauts` involves a finite number of dual-tableau steps, whereas definition `Itauti` involves a dual-tableau with only finite paths.

The definition of `Itauti` was easier to work with than that of `Itauts`, avoiding our difficult earlier proof of Lemma 17. However, using both definitions and proving their equivalence (in the case of finitely branching rules) essentially shows that, even when using the definition `Itauti`, we need only finitely many steps (which is implicit in the way `Itauts` is defined).

We then obtained further necessary results, such as the monotonicity of `Itauti` and that I-tautologous is a property of finite character (i.e. whether a set is I-tautologous depends on whether its finite subsets are):

Lemma 20 (Itauti_idt_mono). *Every superset of an I-tautologous set is I-tautologous: for every* `s`*, if* `s` *is I-tautologous w.r.t. the set* `idt_tab_rule` *of dual-tableau rules then so is every superset* `t` *of* `s`*:*

```
 !s. Itauti idt_tab_rule s
                ==> !t. s SUBSET t ==> Itauti idt_tab_rule t
```

Lemma 21 (ITAUTI_IDT_FINITE). *Every I-tautologous set* `s` *has a finite I-tautologous subset* `t`*:*

```
!s. Itauti idt_tab_rule s ==>
           ?t. FINITE t /\ t SUBSET s /\ Itauti idt_tab_rule t
```

At this point we recall that a tableau branch leaf is closed if it contains signed formulae (F, X) and (T, X) for some formula X. A leaf is atomically closed if it contains (F, p) and (T, p) for some atomic formula p (in our encoding, $p =$ `Atom` a for some atom a). Fitting [Fit17, Definition 2] uses atomic closure in his completeness proofs but we have used closure without this restriction. We want to examine whether this makes any real difference.

Having defined `Itauti` in Definition 23, we now define a generalised version `Itautg`, which is like `Itauti` except that it allows us to specify the requirement for a leaf to be considered closed (thus `Itauti` = `Itautg br_closed`).

Definition 24 (`Itautg`). *For every predicate `cl` on signed formula sets and every rule set `rs`, a set `top` of signed formulae satisfies `Itautg cl rs` iff*

(i) `top` satisfies `cl`, or

(ii) some rule `(top, rb)` in `rs` is applicable to `top` to obtain the conclusion `rb` and every resulting branch `br` in `rb` is I-tautologous w.r.t. `cl` and `rs`

and `Itautg cl rs` is the unique minimal predicate (set) such that (i) and (ii) hold.

```
(!top. cl top ==> Itautg cl rs top) /\
(!top. (?rb. (top,rb) IN rs /\ !br. br IN rb ==> Itautg cl rs br)
                        ==> Itautg cl rs top)
```

Then `Itautg at_closed idt_tab_rule` is the set of I-tautologous sets, defined in terms of atomic closure, for intuitionistic dual-tableaux.

We then showed that a closed dual-tableau can be extended to an atomically closed dual-tableau, so that requiring dual-tableau closure to be atomic makes no difference: this justifies our approach to simplify proofs by not working throughout in terms of atomic closure.

Lemma 22 (`atomic_closure`). *A set is I-tautologous (per Definitions 23 and 13) iff it is I-tautologous (defined to require atomic closure):*

```
Itauti idt_tab_rule sfs <=> Itautg at_closed idt_tab_rule sfs
```

We now discuss the three assumptions which we did not incorporate in our definition of I-tautologous:

(i) Finite character: Fitting defines that an I-tautologous set S must have a finite I-tautologous subset. Our Definition 23 does not require this, but we proved, in Lemma 21, that it holds as a consequence. Our I-tautologous sets are built from dual-tableaux, and each such dual-tableau is a finite structure. Thus if the root $\{S\}$ of the dual-tableaux contains an infinite set S of signed formulae, then we can be assured that our finite dual-tableau will "touch" only a finite subset of its members. Indeed, this is essentially the reason why Lemma 21 holds.

(ii) Atomic closure: We dropped this assumption and allowed closure on arbitrary formulae as it made our task easier. As discussed above, we have since gone back and proved Lemma 22 that everything also goes through if we demand atomic closure. Essentially, this required us to prove that a dual-tableau which is closed using non-atomic closure can be extended to a dual-tableau which is closed atomically.

(iii) Single use restriction: This restriction is also redundant: in fact, by inspection, it can be seen that applying a rule to an inactive formula does not make progress towards a closed dual-tableau. This is noted by Fitting when he observes that "Dual tableaus are sound and complete with or without a single use restriction, but a single use restriction is better for proof search. Indeed, it easily gives us decidability." That each formula is "principal" only once is also redundant as already stated by Fitting [Fit17, just after his Definition 2].

6.2 The Lindenbaum construction

We now discuss proving Fitting's "after Lindenbaum" theorem [Fit17, Theorem 1]. Fitting assumes that the set of signed formulae is countable. We proved the general lemma which expresses the effect obtained from the Lindenbaum construction.

Definition 25 (`maxnon`)**.** *`maxnon` P s means that the set s does not satisfy the predicate P, but that every proper superset of s satisfies P.*

```
maxnon : ('a set -> bool) -> 'a set -> bool
maxnon_def : maxnon P s = ~ P s /\ !t. s PSUBSET t ==> P t
```

Here, `PSUBSET` captures $s \subset t$ (the proper subset relation).

Definition 26 (`ctns1`)**.** *`ctns1` cs m means that the set m contains at least one member of the set of sets cs.*

```
ctns1 : ('a set set) -> 'a set -> bool
ctns1_def : ctns1 cs m <=> ?c. c IN cs /\ c SUBSET m
```

Lemma 23 (`MAXNON_CTNS1`)**.** *Provided that we are dealing with members of a countable set U, if `cs` is a set of finite subsets of U, `m` $\subseteq U$, and `m` does not contain any member of `cs`, then there exists a set `s` $\subseteq U$ which is a superset of `m` and does not contain any member of `cs` and is maximal with that property.*

```
countable (UNIV : 'a set)
==> (cs : 'a set set) SUBSET FINITE  ==> ~ (ctns1 cs (m : 'a set))
==> ?s : 'a set. m SUBSET s /\ maxnon (ctns1 cs) s
```

Here, we take U to be the set of all members of its type, so U is `UNIV`, the set of all things (of the type in question), and then `m` $\subseteq U$, `s` $\subseteq U$ and, for $c \in$ `cs`, $c \subseteq U$ hold automatically, which is why they do not appear explicitly in the encoding but do appear in the plain text.

From `ITAUTI_IDT_FINITE` and `MAXNON_CTNS1` we proved the following lemma.

Lemma 24 (`LINDENBAUM_I`)**.** *Provided that the set of all signed-formulae is countable, if `s` is not I-tautologous then `s` has a superset `M` which is maximal non-I-tautologous:*

```
countable (UNIV : 'a sf set)
 ==> ~ (Itauti idt_tab_rule s)
 ==> ?M : 'a sf set. s SUBSET M /\ maxnon (Itauti idt_tab_rule) M
```

To use this result, we prove that the set of signed-formulae is countable as follows.

Lemma 25 (FORMULAE_COUNTABLE, SF_COUNTABLE). *If the set `UNIV : 'a set` of all atoms is countable then the set `UNIV : 'a formula set` of all formulae (built from those atoms) is countable, as is the set `'a sf set` of all signed-formulae.*

```
countable (UNIV : 'a set) ==> countable (UNIV : 'a formula set)
countable (UNIV : 'a set) ==> countable (UNIV : 'a sf set)
```

A simple way to ensure that the set of atomic formulae is countable is to assume that they are indexed by the natural numbers: for example, as the infinite set $p_0, p_1, p_2, \cdots$. In HOL4, we can achieve our goal by specifying that the type variable `'a` in the type `'a sf` of signed formulae is, in fact, the type `num` of natural numbers. We thus obtain

Lemma 26 (LINDENBAUM). *Assume that the atomic formulae are indexed by the natural numbers: that is, let `'a` be `num` in `'a sf`. Then, if `s` is non-I-tautologous, then `s` has a superset `M` which is maximal non-I-tautologous.*

```
~ (Itauts idt_tab_rule s) ==>
?M : num sf set. s SUBSET M /\ maxnon (Itauts idt_tab_rule) M
```

Here, we specify the type of `M` as `num sf set` which causes `'a sf` to be instantiated to `num sf`. That is, `num sf` is `bool # num formula`: see Section 3.1.

6.3 The canonical model, Truth Lemma and completeness

The canonical model is built out of a (non-empty) set of "worlds" built from maximal non-I-tautologous sets [Fit17, just above Theorem 3]. We therefore define a new type `worlds` representing the set of maximal non-I-tautologous sets. But first, we have to show that this set is non-empty, because types in HOL4 are non-empty.

Lemma 27 (EX_NON_TAUT). *If the atomic formulae are indexed by the natural numbers then there is a maximal non-I-tautologous set of signed formulae.*

```
?(M :num sf set). maxnon (Itauts idt_tab_rule) M
```

Definition 27. *The new type `worlds` is isomorphic to the set of maximal-non-I-tautologous sets.*

```
val worlds_TY_DEF = new_type_definition ("worlds", EX_NON_TAUT) ;
```

That is, we define the new type `worlds` to be isomorphic to the set of things satisfying the property `maxnon (Itauts idt_tab_rule)`: namely the set of maximal non-I-tautologous sets which we have just shown to be non-empty by Lemma 27. The function `new_type_definition` also creates functions and a theorem expressing this isomorphism.

Lemma 28 (worlds_abs_rep). *Assuming the atomic formulae are indexed by the natural numbers, there exists a function `w_rep` from `worlds` to maximal non-I-tautologous sets and a function `w_abs` from sets of signed formulae to `worlds` such that:*

(i) for every world `a` *of type* `worlds`*,* `w_abs (w_rep a) = a`*; and*

(ii) for every set `s` *of signed formulae* `w_rep (w_abs s) = s` *iff* `s` *is maximal non-I-tautologous wrt.* `idt_tab_rule`*.*

```
w_rep : worlds -> num sf set
w_abs : (num sf set) -> worlds

 ( !(a :worlds). w_abs (w_rep a) = a ) /\
    ( !(s :num sf set). (w_rep (w_abs s) = s) <=>
        maxnon (Itauts (idt_tab_rule :(num sf set) rule set)) s )
```

Here, we specify that the atomic formula are indexed by the natural numbers by setting the type of s to be `num sf set`.

We now have a set of worlds built out of maximal non-I-tautologous sets of signed formulae. We define the canonical model over these worlds by defining the valuation of atoms over these worlds and the binary Kripke relation between worlds.

Definition 28 (`at_val, idt_R`)**.** *The truth value* `at_val` *of an atomic formula* `Atom a` *at a world* `w` *is true iff* `(F, Atom a)` *is in the set* `w`*. The world Δ is an* `idt_R`*-successor of the world Γ iff $\{f \mid (F, f) \in \Gamma\} \subseteq \{f \mid (F, f) \in \Delta\}$.*

```
at_val w a = (F, Atom a) IN w_rep w
idt_R gamma delta =
   (FST (mk_seq (w_rep gamma)) SUBSET FST (mk_seq (w_rep delta)))
```

Here, the isomorphism function `wrep` provided by `HOL4` identifies a world `delta` with its corresponding set `w_rep delta` of signed-formulae, and similarly for world `gamma`. We then "partition" the F-signed formulae from the T-signed formulae from these sets of signed formulae by turning each into the sequents $Fs_\Gamma \vdash Ts_\Gamma$ and $Fs_\Delta \vdash Ts_\Delta$, respectively, using `mk_seq`. Projecting onto the first component of these sequents gives us Fs_Γ and Fs_Δ, respectively, and the `SUBSET` construct then gives us the desired result.

The canonical model is thus built from `worlds`, `at_val` and `idt_R` in the usual way and we need to prove the Truth Lemma. For proving the Truth Lemma, we proved

Lemma 29 (`NON_ITAUT_RULE`)**.** *If the rules from the rule set* `rs` *are finitely branching, and* `s` *is maximal non-I-tautologous w.r.t.* `rs`*, and all extensions by a context of the skeleton rule (*`top/bot`*) are contained in* `rs`*, then if* `top` *is in* `s` *then so is some member of* `bot`*.*

```
IMAGE SND rs SUBSET FINITE
     ==> maxnon (Itauts rs) s
     ==> is_tab_rule (top, bot) SUBSET rs
     ==> top IN s  ==> ?br. br IN bot /\ br SUBSET s
```

Assume the canonical model is built from `worlds`, `at_val` and `idt_R` in the usual way using Definition 28, thus giving rise to a forcing relation `forces idt_R at_val` which maps a particular world Γ and a particular formula X to true or false. The following result corresponds to Fitting's "Intuitionistic Truth Lemma" [Fit17, Theorem 3]. It is proved by induction on the formula X, using Lemma 29.

Lemma 30 (TRUTH_LEMMA). *For all formulae X and for all worlds Γ (ie, maximal non-I-tautologous sets of signed formulae)*

(i) if (T, X) in Γ then Γ does not force X, and

(ii) if (F, X) in Γ then Γ does force X:

```
!X gamma.
 ((T, X) IN w_rep gamma ==> ~ (forces idt_R at_val gamma X)) /\
 ((F, X) IN w_rep gamma ==>   (forces idt_R at_val gamma X))
```

Again, we utilise the isomorphism function `w_rep` to find the set of signed formulae represented by Γ.

For the completeness theorem, we first state a lemma about the canonical model.

Lemma 31 (idt_complete). *In the canonical model, if every world w forces formula f then the singleton signed formula set $\{(T, f)\}$ is I-tautologous.*

```
(!w. forces idt_R at_val w f) ==> Itauts idt_tab_rule {(T,f)}
```

Now, using the contrapositive form, we get completeness as desired:

> if no dual-tableau for the set $\{(T, f)\}$ is closed then f is falsifiable in some Kripke model [Fit17].

Theorem 2 (idt_complete_cp). *If the singleton signed formula set $\{(T, f)\}$ is not I-tautologous (ie. the formula f has no dual-tableau proof), then there is a world in the canonical model which does not force f.*

```
~ Itauts idt_tab_rule {(T,f)} ==> ?w. ~ forces idt_R at_val w f
```

Proof. For a formula f, if $\{(T, f)\}$ has no closed dual-tableau, that is, if $\{(T, f)\}$ is not I-tautologous, then by Lemma 26, it is contained in a maximal non-I-tautologous set Γ, which is a world in the canonical model. Then, by Lemma 30, $\Gamma \not\Vdash f$. □

6.4 Relaxing the countable constraint

The proof described above required that the set of formulae is countable: proving that this holds, if the set of atoms is countable, was not trivial (see Lemma 25). An alternative is to drop this requirement and to use Zorn's lemma, which is provided in HOL4, giving a version of Lemma 23 without the countable set restriction.

Lemma 32 (MAXNON_CTNS1_ZORN). *If cs is a set of finite sets, and m does not contain any member of cs, then there exists an s which is a superset of m and does not contain any member of cs and is maximal w.r.t. that property.*

```
cs SUBSET FINITE
   ==> ~ (ctns1 cs m)
   ==> ?s : 'a set. m SUBSET s /\ maxnon (ctns1 cs) s
```

Both these approaches require the finite character property of a set being I-tautologous: that is, that an I-tautologous set has an I-tautologous finite subset.

Finite characterisation of being I-tautologous is conceptually easy, as discussed earlier, and proved in Lemma 21. However another approach here is to define an I-tautologous set as one which has a finite I-tautologous subset, as Fitting does, in [Fit17, Definition 7]. We did this (calling it `fITauts`), which made it easy to prove analogues of the results Lemmas 20 and 21, but other things become more difficult. For example, we proved (at quite some length) this analogue of Lemma 29. Note that, compared with Lemma 29, we proved it only specifically for the set of rules for intuitionistic dual-tableaux.

Lemma 33 (NON_FITAUT_RULE). *If s is maximal non-I-tautologous wrt. the rules* `idt_tab_rule` *for intuitionistic dual-tableaux, and the extensions by a context of the skeleton rule* `(top/bot)` *are contained in* `idt_tab_rule`*, then if* `top` *is in s, then so is some member of* `bot`

```
maxnon (fItauts idt_tab_rule) (s : 'a sf set)
    ==> is_tab_rule (top, bot) SUBSET idt_tab_rule
    ==> top IN s ==> ?br. br IN bot /\ br SUBSET s
```

We didn't pursue this approach further, and Lemma 21 makes it rather redundant. It really just illustrates that until one actually performs the proofs, one doesn't really know which approach will be simplest to prove.

7 Conclusions

We have shown how to encode the meta-theory of dual-tableaux for intuitionistic logic into `HOL4`. In the process, we have verified all of the theorems provided by Melvin Fitting in his chapter in this volume, although our proofs sometimes proceed differently. We have also highlighted how inductive definitions often make proofs easier since we can perform structural induction on the clauses that make up the inductive definition. All of our `HOL4`-code can be found via the link (`http://users.cecs.anu.edu.au/~jeremy/hol/idt/`), and is also available on GitHub at `https://github.com/jeremydaw/idt` in the directory `hol`.

Regarding the effort required. The proof script is 2100 lines of `HOL4`-code. Contrasted against Fitting's original chapter [Fit17], this is a similar length — but containing much more detail of small proof steps, and much less descriptive and explanatory material. This contains some results which were proved in a roundabout way, or with some duplication of effort (such as the issue of `Itauti` versus `Itauts`, see §6.1.3), and a small amount of theory not specific to this particular task, such as the proof of Lemma 25. (Generally HOL offers good support for most common generic reasoning tasks, although not for proving an algebraic data type to be countable).

One caveat: Jeremy Dawson has over 20 years of experience in interactive theorem proving, and yet it took him 2 months of full-time work to complete these proofs, so interactive theorem proving is time-consuming and laborious!

Acknowledgements. We are grateful to the anonymous reviewers for their suggestions for improvements.

References

[Bet53] E W Beth. On Padoa's method in the theory of definition. *Indag. Math.*, 15:330–339, 1953.

[DG02] Jeremy E. Dawson and Rajeev Goré. Formalised cut admissibility for display logic. In *TPHOLs02: Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics*, volume LNCS 2410:131–147. Springer, 2002.

[DG10] Jeremy E. Dawson and Rajeev Goré. Generic methods for formalising sequent calculi applied to provability logic. In LPAR-17: *Proceedings of the International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LNCS 6397:263–277, 2010.

[Fit83] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library*. D. Reidel, Dordrecht, Holland, 1983.

[Fit17] Melvin Fitting. Tableaus and dual tableaus. In *Ewa Orlowska Volume*. 2017.

[Gor08] Mike Gordon. Twenty years of theorem proving for HOLs past, present and future. In TPHOLs 2008 *Proceedings of Theorem Proving in Higher Order Logics, 21st International Conference*, LNCS 5170:1–5, 2008.

[Kri59] Saul Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24(1):1–14, March 1959.

[OGP11] Ewa Orlowska and Joanna Golińska-Pilarek. *Dual Tableaux: Foundations, Methodology, Case Studies*, volume 33 of *Trends in Logci*. Springer, 2011.

[Sco93] Dana S Scott. A type-theoretical alternative to iswim, cuch, owhy. *Theoretical Computer Science*, 1993.