Satisfiability Of Connected Contact Logics

Anton Dudov

2019-9-28

Contents

1 Tableau Method For Classic Propositional Logic

1.1 What Is A Tableau?

A tableau method is a formal proof procedure. First, it could be used as a refutation procedure: to show a formula X is valid we begin with some syntactical expression intended to assert it is not. This expression is broken down syntactically, generally splitting things into several cases. This part of a tableau procedure - the tableau expansion stage - can be thought of as a generalization of disjunctive normal form expansion. Generally, it moves from formulas to subformulas. Finally, there are rules for closing cases: impossibility conditions based on syntax. If each case closes, the tableau itself is said to be closed. A closed tableau beginning with an expression asserting that X is not valid is a tableau proof of X.

There is a second way of thinking about the tableau method: as a search procedure for models meeting certain conditions. Each branch of a tableau can be considered to be a partial description of a model. In automated theorem-proving, tableaus can be used to generate counter-examples.

The connection between the two roles for tableaus - as a proof procedure and as a model search procedure - is simple. If we use tableaus to search for a model in which X is false, and we produce a closed tableau, no such model exists, so X must be valid.

1.2 Classical Propositional Tableaus

We will look into the signed tableau system for classical propositional logic.

First, we need syntactical machinery for asserting the invalidity of a formula, and for doing case analysis. For this purpose two signs are introduced: *T* and *F*, where these are simply two new symbols, not part of the language of formulas. *Signed formulas* are expressions of the form FX and TX, where X is a formula. The intuitive meaning of FX is that X is *false* (in some model). Similarly, TX intuitively asserts that X is *true*. Then FX is the syntactical device for (informally) asserting the invalidity of X. A tableau proof of X begins with FX.

Next, we need machinery (rules) for breaking signed formulas down and doing a case division. We will define rules for each logical operator ($\neg \wedge \vee \Rightarrow \Leftrightarrow$).

The treatment of **negation** is straightforward: from $T\,\neg X$ we get $F\,X$ and from $F\,\neg X$ we get $T\,X$. These rules can be conveniently presented as follows.

$$\frac{T\neg X}{FX} \qquad\qquad \frac{F\neg X}{TX}$$

The rules for **conjunction** are somewhat more complex. From truth tables we know that if $X \wedge Y$ is *true*, X must be *true* and Y must be *true*. Likewise, if $X \wedge Y$ is *false*, eigher X is *false* or Y is *false*. This involves a split into two cases. Corresponding syntactic rules are as follows.

$$\frac{TX \wedge Y}{\begin{array}{c}TX\\TY\end{array}} \qquad\qquad \frac{FX \wedge Y}{FX \mid FY}$$

The rules for **disjunction** are similar. From truth tables we know that if $X \vee Y$ is *true*, eigher X is *true* or Y is *true*. This involves a split into two cases. Likewise, if $X \vee Y$ is *false*, X must be *false* and Y must be *false*. Corresponding syntactic rules are as follows.

$$\frac{TX \vee Y}{TX \mid TY} \qquad\qquad \frac{FX \vee Y}{\begin{array}{c}FX\\ FY\end{array}}$$

The rules for **implication**. From truth tables we know that if $X \Rightarrow Y$ is *true*, eigher X is *false* or Y is *true*. Likewise, if $X \Rightarrow Y$ is *false*, X must be *true* and Y must be *false*. Corresponding syntactic rules are as follows.

$$\frac{TX \Rightarrow Y}{FX \mid TY} \qquad\qquad \frac{FX \Rightarrow Y}{\begin{array}{c}TX\\ FY\end{array}}$$

The rules for **equivalence**. From truth tables we know that if $X \Leftrightarrow Y$ is *true*, eigher X is *true* and Y is *true* or X is *false* and Y is *false*. Likewise, if $X \Rightarrow Y$ is *false*, eigher X is *true* and Y is *false* or X is *false* and Y is *true*. Corresponding syntactic rules are as follows.

$$\frac{TX \Leftrightarrow Y}{\begin{array}{c|c}TX & FX\\ TY & FY\end{array}} \qquad\qquad \frac{FX \Leftrightarrow Y}{\begin{array}{c|c}TX & FX\\ FY & TY\end{array}}$$

The standard way of displaying tableaus is as downward branching trees with signed formulas as node labels. Indeed, the tableau method is often referred to as the tree method. Think of a tree as representing the disjunction of its branches, and a branch as representing the conjunction of the signed formulas on it.

When using a tree display, a tableau expansion is thought of temporally, and one talks about the stages of constructing a tableau, meaning the stages of growing a tree. The rules given above are thought of as branch-lengthening rules. Thus, a branch containing $T \neg X$ can be lengthened by adding a new node to its end, with $F\,X$ as a label. Likewise a branch containing $F\,X \vee Y$ can be lengthened with two new nodes, labelled $F\,X$ and $F\,Y$ (take the node with $F\,Y$ as the child of the one labelled $F\,X$). A branch containing $T\,X \vee Y$ can be split - its leaf is given a new left and a new right child, with one labelled $T\,X$, the other $T\,Y$. This is how the schematic rules above are applied to trees.

An important point to note is that the tableau rules are non-deterministic. They say what can be done, not what must be done. At each stage, we choose a signed formula occurrence on a branch and apply a rule to it. Since the order of choice is arbitrary, there can be many tableaus for a single signed formula.

Here is the final stage of a tableau expansion beginning with the signed formula $F(X \wedge Y) \Rightarrow (\neg X \wedge \neg Y)$.

1. $F(X \wedge Y) \Rightarrow \neg(\neg X \vee \neg Y)$
2. $TX \wedge Y$
3. $F\neg(\neg X \vee \neg Y)$
4. TX
5. TY

$$\begin{tikzcd}
& 6.\ T\neg X \vee \neg Y \arrow[dl, dash] \arrow[dr, dash] & \\
7.\ T\neg X & & 8.\ T\neg Y \\
9.\ FX & & 10.\ FY
\end{tikzcd}$$

In this we have added numbers for reference purposes. Items 2 and 3 are from 1 by $F \Rightarrow$. 4 and 5 are from 2 by $T\wedge$. 6 is from 3 by $F\neg$. 7 and 8 are from 6 by $T\vee$. 9 is from 7 by $T\neg$. 10 is from 8 by $T\neg$.

Finaly, the conditions for closing off a case (declaring a branch closed) are simple. A **branch is closed** if it contains a contradiction, i.e. TA and FA for some formula A. A **branch is opened** if it does not contain any contradiction. If each branch is closed, then the **tableau is closed**. A closed tableau for FX is a tableau proof of X, meaning that A is a tautology. The tableau displayed above is closed, so the formula $(X \wedge Y) \Rightarrow (\neg X \wedge \neg Y)$ has a tableau proof.

It may happen that no tableau proof is forthcoming, and we can think of the tableau construction as providing us with contraexamples. Consider the following attempt to prove $(X \Rightarrow Y) \Rightarrow ((Y \Rightarrow X) \Rightarrow \neg Y)$

1. $F((X \Rightarrow Y) \Rightarrow ((Y \Rightarrow X) \Rightarrow \neg Y))$
2. $TX \Rightarrow Y$
3. $F((Y \Rightarrow X) \Rightarrow \neg Y)$
4. $TY \Rightarrow X$
5. $F\neg Y$

$$\begin{tikzcd}
& & & 6.\ TY \arrow[dll, dash] \arrow[drr, dash] & & & \\
& 7.\ FX \arrow[dl, dash] \arrow[dr, dash] & & & & 8.\ TY \arrow[dl, dash] \arrow[dr, dash] & \\
9.\ FY & & 10.\ TX & & 11.\ FY & & 12.\ TX
\end{tikzcd}$$

Item 2 and 3 are from 1 by $F \Rightarrow$, as are 4 and 5 from 3. Item 6 is from 5 by $F\neg$. Items 7 and 8 are from 2 by $T \Rightarrow$, as are 9 and 10 from 4. Items 11 and 12 are also from 4 by $T \Rightarrow$. The leftmost branch is closed because of 6 and 9. The left-right branch is closed because of 7 and 10. The right-left branch is closed because of 8 and 11. But the rightmost branch is not closed. Notice that every non-atomic signed formula has had a rule applied to it on this branch and there is nothing left to do. (For clasical propositional logic it is sufficient to apply a rule to a formula on a branch only once.) In fact the branch yields a counterexample, as follows. Let v be a propositional valuation that maps X to *true* and Y to *true* in accordance to 8 and 12. Now, we work our way back up the branch. Since $v(Y) = true$, $v(\neg Y) = false$, item 5. From $v(X) = true$ follows that $v(Y \Rightarrow X) = true$, item 4. From $v(Y) = true$ follows that $v(X \Rightarrow Y) = true$, item 2. Since $v(Y \Rightarrow X) = true$ and $v(\neg Y) = false$ we have $v((Y \Rightarrow X) \Rightarrow \neg Y) = false$, item 3. Finally, $v((X \Rightarrow Y) \Rightarrow ((Y \Rightarrow X) \Rightarrow \neg Y)) = false$, item 1.

Later, we are going to use the tableau method to produce a model in which the initial formula is valid.

From a different point of view, we can think of a classical tableau simply as a set of sets of signed formulas: a tableau is the set of its all branches, and a branch is the set of signed formulas that occur on it. Semantically, we think of the outer set as the disjunction of its members, and these members, the inner sets, as conjunctions of signed formulas they contain. Considered this way, a tableau is a generalization of disjunctive normal form (a generalization because of formulas more complex than literals can occur). Now, the tableau construction process can be thought of as a variation of the process of converting a formula into a disjunctive normal form.

2 Contact Logics

2.1 Syntax

The language of contact logic consist of:

- *Boolean variables* (a denumerable set $\mathcal{V}$)
- *Boolean constants*: 0 and 1
- *Boolean operations*:
  - $\sqcap$ boolean meat
  - $\sqcup$ boolean join
  - * boolean complement
- *Boolean terms* (or simply *terms*)
- *Propositional connectivies*: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$
- *Propositional constants*: $\top$ and $\bot$
- *Modal connectives*: $\leq$(part-of) and C(contact)
- *Complex formulas* (or simply *formulas*)

Terms are defined in the following inductive process:

- Each Boolean variable is a term
- Each Boolean constant is a term
- If a is a term then a^* is a term
- If a and b are terms then $a \sqcap b$ and $a \sqcup b$ are terms

Atomic formulas are of the form $a \leq b$ and aCb, where a and b are terms.
Formulas are defined in the following inductive process:

- Each propositional constant is a formula
- Each atomic formula is a formula
- If ϕ is a formula then $\neg\phi$ is a formula
- If ϕ and ψ are formulas then $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \Rightarrow \psi)$ and $(\phi \Leftrightarrow \psi)$ are formulas

Abbreviations: $a = b \overset{\text{def}}{=} (a \leq b) \wedge (b \leq a)$, $a \nleq b \overset{\text{def}}{=} \neg(a \leq b)$, $a \neq b \overset{\text{def}}{=} \neg(a = b)$

2.2 Relational semantics

Let $\mathcal{F}$ = (W, R) be a relational system with W $\neq \emptyset$ and R $\subseteq W^2$. We call such systems *frames*. Following Galton[TODO] we may give a spatial meaning of frames naming the elements of W, *cells* and the relation R, *adjacency relation*. Then $\mathcal{F}$ is called *adjacency space*. An example of adjacency space is the chess-board table, the cells are the squares, and two squares are adjacent if they have a common point.

Originally Galton assumed R to be a reflexive and symmetric relation but it is more natural for R to be an arbitrary relation. *Regions* in an adjacency space are arbitrary subsets of W and two sets a and b are in *contact* (aC_Rb) if for some $x \in a$ and $y \in b$ we have xRy. Another way to define this relation is the following. For a subset $a \subseteq W$ define as in modal logic $\langle R\rangle a = \{x \in W : (\exists y \in W)(xRy \text{ and } y \in a)\}$. Then aC_Rb iff $a \cap \langle R\rangle b \neq \emptyset$ iff $\langle R^{-1}\rangle a \cap b \neq \emptyset$, where R^{-1} is the converse relation of R. Note that if R is a symmetric relation then aCb iff $a \cap \langle R\rangle b \neq \emptyset$ iff $\langle R\rangle a \cap b \neq \emptyset$.

Deffinition 1. *By a* ***valuation*** *of the Boolean variables in $\mathcal{F}$ we mean any function $v : \mathcal{V} \to \mathcal{P}(W)$ assigning to each Boolean variable b a subset $v(b) \subseteq W$. The valuation v is then extended inductively to all Boolean terms as follows:*

- $v(0) = \emptyset$
- $v(1) = W$
- $v(a \sqcap b) = v(a) \cap v(b)$
- $v(a \sqcup b) = v(a) \cup v(b)$
- $v(a^*) = W \setminus v(a) = -v(a)$

Deffinition 2. *The pair $\mathcal{M} = (\mathcal{F}, v)$ is called* **model**. *The truth of a formula ϕ in $\mathcal{M}$ ($\mathcal{M} \models \phi$ or $\mathcal{F}, v \models \phi$) is extended inductively to all Boolean terms as follows:*

- *For atomic formulas:*
  - $\mathcal{M} \models \top$
  - $\mathcal{M} \not\models \bot$
  - $\mathcal{M} \models a \leq b \iff v(a) \subseteq v(b)$
  - $\mathcal{M} \models aCb \iff v(a)\ C_R\ v(b \iff (\exists x \in v(a))(\exists y \in v(b))(xRy)$
- *For complex formulas:*
  - $\mathcal{M} \models \neg\phi \iff \mathcal{M} \not\models \phi$
  - $\mathcal{M} \models \phi \wedge \psi \iff \mathcal{M} \models \phi$ *and* $\mathcal{M} \models \psi$
  - $\mathcal{M} \models \phi \vee \psi \iff \mathcal{M} \models \phi$ *or* $\mathcal{M} \models \psi$
  - $\mathcal{M} \models \phi \Rightarrow \psi \iff \mathcal{M} \not\models \phi$ *or* $\mathcal{M} \models \psi$
  - $\mathcal{M} \models \phi \Leftrightarrow \psi \iff$ *(*$\mathcal{M} \models \phi$ *and* $\mathcal{M} \models \psi$*) or (*$\mathcal{M} \not\models \phi$ *and* $\mathcal{M} \not\models \psi$*)*

Let us note that in the above semantics we evaluate formulas not locally at points[TODO ??], as it is in the standard modal semantics, but globally in the whole model and this is one of the main differences of the present modal approach with the standard Kripke approach.

Deffinition 3. *A model $\mathcal{M}$ is a **model of a formula** ϕ if ϕ is true in $\mathcal{M}$.*

Deffinition 4. *If ϕ has a model $\mathcal{M}$, then ϕ is **satisfiable**.*

Deffinition 5. *$\mathcal{M}$ is a **model of a set of formulas** A if $\mathcal{M}$ is a model of all formulas from A.*

Deffinition 6. *A formula ϕ **is true** (or **valid**) in a frame $\mathcal{F}$ ($\mathcal{F} \models \phi$), if $\mathcal{M} \models \phi$ for all models $\mathcal{M}$ based on $\mathcal{F}$, i.e. for all valuations v we have $\mathcal{F}, v \models \phi$.*

Lemma 7. *(Equality of formulas) Let f and g are formulas. Then*
$f = g \implies v(f) = v(g)$.

Lemma 8. *(Equality of terms) Let a and b are terms. Then*
$a = b \implies v(a) = v(b)$.

Lemma 9. *(Zero term) Let a and b are terms. Then*
$a \leq b \implies a \sqcap b^* = 0$

Lemma 10. *(Non-zero term) Let a and b are terms. Then*
$\neg(a \leq b) \implies a \sqcap b^* \neq 0$

TODO: axiom or lemma or neither??

Axiom 11. *(Reflexivity) Let a be a term. Then*
$a \neq 0 \implies aCa$.

Axiom 12. *(Symmetry) Let a and b are terms. Then*
$aCb \iff bCa$.

Lemma 13. *(Monotonicity) Let a and b are terms. Then*
$aCb \wedge a \leq a' \wedge b \leq b' \implies a'Cb'$.

Lemma 14. *(Distributivity) Let a and b are terms. Then*
$aC(b \sqcup c) \iff aCb \vee aCc$, *TODO what is the other one?* $aC(b \sqcup c) \iff aCb \vee aCc$.

Lemma 15. *Let a, b, c are terms and f, g are formulas. The following formulas are true:*

- $f \wedge T \implies f, T \wedge f \implies f$
- $f \wedge F \implies F, F \wedge f \implies F$
- $f \vee T \implies T, T \vee f \implies T$
- $f \vee F \implies f, F \vee f \implies f$

- $a \sqcap 0 \implies 0, 0 \sqcap a \implies 0$
- $a \sqcup 0 \implies a, 0 \sqcup a \implies a$
- $a \sqcap 1 \implies a, 1 \sqcap a \implies a$
- $a \sqcup 1 \implies 1, 1 \sqcup a \implies 1$
- $(a \sqcup b)Cc \iff aCc \vee bCc$
- $(a \sqcup b) \leq c \iff a \leq c \wedge b \leq c$
- $aCb \implies a \neq 0 \wedge b \neq 0$
- $a \sqcap b \neq 0 \implies aCb$
- $a = 0 \vee b = 0 \implies \neg(aCb)$
- $0 \leq a \implies T$
- $a \leq 1 \implies T$
- $0C0 \implies F$
- $aC0 \implies F$
- $1C1 \implies T$
- $aC1 \implies a \neq 0$
- $a \neq 0 \implies aCa$

2.3 Formula satsfiability

Let ψ be a propositional formula. Let us build a tableau beginning with ψ. If the tableau has an opened branch then ψ is satisfiable. Unfortunately, for the contact logic this is not enough because we need to verify the modal connectives ($\leq$ and C).

Let ϕ be a formula. Let us build a tableau beginning with ϕ. Let the tableau has an opened branch B. The branch B is a set of signed atomic formulas of the following type:

- $TC(a, b)$
- $FC(e, f)$
- $Ta \leq b$
- $Fa \leq b$

where a and b are terms.
B is an opened branch, so there are no contradicting formulas in it, i.e. $(\neg\exists X)(TX \in B \wedge FX \in B)$. The signed atomic formulas could be written as atomic formulas as follows:

- $TC(a, b) \to C(a, b)$ (contact)
- $FC(a, b) \to \neg C(a, b)$ (non-contact)
- $Ta \leq b \to a \leq b \to a \sqcap b^* = 0 \to g = 0$ (zero term)
- $Fa \leq b \to \neg(a \leq b) \to a \sqcap b^* \neq 0 \to d \neq 0$ (non-zero term)

where a,b,d and g are terms.

All atomic formulas in the branch should be satisfied, so we can think of it as a conjunction of them. Let's call it a **branch conjunction**. It is sufficient to build a satisfiable model for the branch conjuction to verify that ϕ is satisfiable. Building such a model could be done a lot more effective than building a model for an arbitrary formula because it is just a conjuction.

Deffinition 16. *Let ϕ be a formula. Let T be a tableau beginning with ϕ. Let B be a set of all atomic signed formulas in a branch of T, as follows:*

$$\begin{aligned} B = \{TC(a_i, b_i) \mid i \in \{1, \ldots, I\}\} \cup \{Fd_j = 0 \mid j \in \{1, \ldots, J\}\} \cup \\ \{FC(e_k, f_k) \mid k \in \{1, \ldots, K\}\} \cup \{Tg_l = 0 \mid l \in \{1, \ldots, L\}\} \end{aligned} \tag{1}$$

A **branch cojnuction** β is the following formula:

$$\bigwedge_{i=1}^{I} C(a_i, b_i) \wedge \bigwedge_{j=1}^{J} d_j \neq 0 \wedge \bigwedge_{k=1}^{K} \neg C(e_k, f_k) \wedge \bigwedge_{l=1}^{L} g_l = 0 \tag{2}$$

2.4 Branch conjunction model building

The branch conjunction formula β is satisfiable if β has a model $\mathcal{M}$. We have to construct such a model $\mathcal{M} = (\mathcal{F}, v) = ((\mathrm{W}, \mathrm{R}), v)$.

Deffinition 17. *A denumerable set of n boolean variables $\mathcal{V}_n$ is defined as follows:*

$$\mathcal{V}_n = \{x_1, x_2, \ldots, x_n\} \tag{3}$$

Let n be the number of the unique boolean variables in β. Let $\mathcal{V}_n$ be the set of them. There are four types of atomic formulas in β, namely contacts, non-contacts, zero terms and non-zero terms. Only contacts and non-zero terms require existance of modal points. The valuation v should assign a set of modal points to each boolean variable $x \in \mathcal{V}_n$. One way to define v and W is the following:

- Assign an empty set of modal points for each boolean variable $x \in \mathcal{V}_n$.
  $v = \{ \langle x, \emptyset \rangle \mid x \in \mathcal{V}_n \}$
- Add a modal point for each term in the contacts and non-zero terms of β.
  $\mathrm{W} = \{ p_i \mid \mathrm{C}(a_i, b_i) \in \beta \} \cup \{ p_{I+i} \mid \mathrm{C}(a_i, b_i) \in \beta \} \cup \{ p_{I*2+j} \mid d_j \neq 0 \in \beta \}$
- Extend v such as:

- $p_i \in \upsilon(a_i)$ for each $\mathrm{C}(a_i, b_i) \in \beta$
- $p_{I+i} \in \upsilon(b_i)$ for each $\mathrm{C}(a_i, b_i) \in \beta$
- $p_{I*2+j} \in \upsilon(d_j)$ for each $d_j \neq 0 \in \beta$

Deffinition 18. *Let $\mathcal{V}_n$ be a denumerable set of n boolean variables. Let W be a set of modal points. Let υ be a valuation over $\mathcal{V}_n$ and W. Then $X_i \subseteq W$ is the evaluation set of the boolean variable x_i in $\mathcal{V}_n$.*

$$X_i = \upsilon(x_i) \ for x_i \in \mathcal{V}_n \tag{4}$$

Deffinition 19. *Let $\mathcal{V}_n$ be a denumerable set of n boolean variables. Let W be a set of modal points. Let υ be a valuation over $\mathcal{V}_n$ and W. Then X is the union of all evaluations of boolean variables in $\mathcal{V}_n$.*

$$X = \bigcup_{x_i \in \mathcal{V}_n} \upsilon(x_i) = \bigcup_{x_i \in \mathcal{V}_n} X_i \tag{5}$$

Let t be an arbitrary term. Let p $\notin$ W be a new modal point. Let us extend υ such as p $\in \upsilon(t)$. By the valuation definition $\upsilon(t)$ is a composition of interesections, unions and compliments of some $X_i \subseteq$ W. Therefore, the modal point p should be added to zero or more X_i depending on the boolean operations in the term t. With n boolean variables there are 2^n ways of adding the point p.

Deffinition 20. *A **variable evaluation** $\mathcal{E}_n$ for n boolean variables is a sequence of 1s and 0s, as follows:*

$$\mathcal{E}_n = < e_1, e_2, \ldots, e_n >, \ where \ e_1, \ldots, e_n \in \{0, 1\} \tag{6}$$

For *n* boolean variables there are 2^n unique variable evaluations. We will define the modal points as variable evaluations. By the definition of the valuation, it is not possible to distinguish two or more different modal points in some subsets. For example, the W $\setminus$ X subset. The modal point representation also have this limitaion. The modal connectives require existence of a point in those sets. It is sufficient to work with only one point from them.

Deffinition 21. *Let $\mathcal{E}_n$ be a variable evaluation for n boolean variables. Then $(\mathcal{E}_n)^i$ is the i-th element in the sequence $\mathcal{E}_n$.*

Deffinition 22. *Let $\mathcal{V}_n$ be a denumerable set of n boolean variables. The set of **all unique variable evaluations** W_n over n variables is defined as follows:*

$$W_n = \{< e_1, e_2, \ldots, e_n > | \ e_1, \ldots, e_n \in \{0, 1\}\} \tag{7}$$

Deffinition 23. *Let $\mathcal{V}_n$ be a denumerable set of n boolean variables. Let W_n be the set of all unique points over n variables. Let $W \subseteq W_n$. Then the **valuation** $\upsilon_n : \mathcal{V}_n \longrightarrow \mathcal{P}(W)$ is inductively defined as follows:*

- *Boolean constants*

- $\upsilon_n(0) = \emptyset$
- $\upsilon_n(1) = W$

- *Boolean variables*
  - $\upsilon_n(x_i) = \{\mathcal{E}_n \mid \mathcal{E}_n \in W \wedge (\mathcal{E}_n)^i = 1\}$, *for* $x_i \in \mathcal{V}_n$

- *Boolean terms*
  - $\upsilon_n(a \sqcap b) = \upsilon_n(a) \cap \upsilon_n(b)$
  - $\upsilon_n(a \sqcup b) = \upsilon_n(a) \cup \upsilon_n(b)$
  - $\upsilon_n(a^*) = W \setminus \upsilon_n(a) = -\upsilon_n(a)$

Let β be a branch conjunction as in 16 . Let W be a set of modal points. Let $\mathcal{V}_n$ be the set of boolean variables in β. Let υ_n be a valuation as in 23 . Let $W \subseteq W_n$. Let R $\subseteq W^2$ be a reflexive and symetric relation.

Deffinition 24. *The zero terms in β are satisfied iff each zero term's valuation is the empty set.*
$$(\forall (g_l = 0) \in \beta)(\upsilon_n(g_l) = \emptyset) \tag{8}$$

Deffinition 25. *The non-contacts in β are satisfied iff:*

$$(\forall_{\neg C(e_k, f_k) \in \beta}) \neg (\exists x \in \upsilon_n(e_k))(\exists y \in \upsilon_n(f_k))(xRy) \tag{9}$$

Lemma 26. *The valuations of the non-contact terms in β does not have a common modal point. Follows from the reflexivity of R.*

$$(\forall_{\neg C(e_k, f_k) \in \beta})(\upsilon_n(e_k) \cap \upsilon_n(f_k) = \emptyset) \tag{10}$$

Deffinition 27. *Let t be an arbitrary boolean term. Then $\mathcal{V}_t$ is the set of the boolean variables used in t.*

Deffinition 28. *Let $\mathcal{V}_n$ be a denumerable set of boolean variables. Then $\mathcal{T}_n$ is the set of all boolean terms with variables of $\mathcal{V}_n$.*

$$\mathcal{T}_n = \{t \mid t \textit{ is a boolean term and } \mathcal{V}_t \subseteq \mathcal{V}_n\} \tag{11}$$

Deffinition 29. *Let $\mathcal{V}_n$ be a denumerable set of boolean variables. Let $\mathcal{T}_n$ be the set of all boolean terms with variables of $\mathcal{V}_n$. Let $W \subseteq W_n$. By a **boolean valuation** of a boolean term in $\mathcal{T}_n$ we mean the function η. It assigns a boolean constant for each pair of term $t \in \mathcal{T}_n$ and variable evaluation $\mathcal{E}_n \in W_n$.*

$$\eta(t, \mathcal{E}_n) \in \{0, 1\} \tag{12}$$

Let $t \in \mathcal{T}_n$. Let $\mathcal{E}_n \in W_n$. The inductive definition of η on the structure of the term t is as follows:

- $\eta(0, E_n) = 0$

- $\eta(1, E_n) = 1$
- $\eta(x_i, E_n) = (E_n)^i$
- $\eta(a \sqcap b, E_n) = \eta(a, E_n) \sqcap \eta(b, E_n)$
- $\eta(a \sqcup b, E_n) = \eta(a, E_n) \sqcup \eta(b, E_n)$
- $\eta(a^*, E_n) = \ ^*\eta(a, E_n)$

Lemma 30. *Let $t \in \mathcal{T}_n$ be an arbitrary term. Let $W \subseteq W_n$. Let $\mathcal{E}_n \in W$. Let η be a boolean valuation. Let υ_n be a valuation as in 23 . By the definition of η and υ_n follows:*

$$\eta(t, \mathcal{E}_n) = 1 \iff \mathcal{E}_n \in \upsilon_n(t) \tag{13}$$

Deffinition 31. *Let $t \in \mathcal{T}_n$ be an arbitrary term. Let η be a boolean valuation. Let $\mathcal{E}_n \in W_n$. Then* $\mathcal{E}_\mathbf{n}$ **satisfies t** *iff $\mathcal{E}_n$ evaluates t to the constant 1.*

$$\mathcal{E}_n\ satisfies\ t \iff \eta(t, \mathcal{E}_n) = 1 \tag{14}$$

Deffinition 32. *Let $t \in \mathcal{T}_n$ be an arbitrary term. Let η be a boolean valuation. The set of all variable evaluations $\mathcal{E}_t$ which satisfies t is defined as follow:*

$$\mathcal{E}_t = \{\mathcal{E}_n \mid \mathcal{E}_n \in W_n \wedge \eta(t, \mathcal{E}_n) = 1\} \tag{15}$$

Deffinition 33. *Let $\mathcal{E}_n \in W_n$. Let $W \subseteq W_n$. Let $W = W \cup \mathcal{E}_n$. Then $\mathcal{E}_n$ is a* **valid modal point of** *β if it does not break the satisfaction of the zero terms and non-contacts in β:*

$$(\forall (g_l = 0) \in \beta)(\mathcal{E}_n \notin \upsilon_n(g_l)) \wedge (\forall_{\neg C(e_k, f_k) \in \beta})(\mathcal{E}_n \notin \upsilon_n(e_k) \cap \upsilon_n(f_k)) \iff$$
$$(\forall (g_l = 0) \in \beta)(\eta(g_l, \mathcal{E}_n) = 0) \wedge (\forall_{\neg C(e_k, f_k) \in \beta})(\eta(e_k, \mathcal{E}_n) = 0 \wedge \eta(f_k, \mathcal{E}_n) = 0)$$

Deffinition 34. *Let $W^v \subseteq W_n$ be the set of all valid modal points of β.*

$$W^v = \{\mathcal{E}_n \mid \mathcal{E}_n \in W_n \wedge \mathcal{E}_n\ is\ a\ valid\ modal\ point\ of\ \beta\} \tag{16}$$

Deffinition 35. *Let x, y $\in W^v$. Then <x, y> is a* **valid connected pair of** *β modal points if it does not break the satisfaction of non-contacts in β.*

$$\begin{aligned}
(\forall_{\neg C(e_k, f_k) \in \beta})\neg((x \in \upsilon_n(e_k) \wedge y \in \upsilon_n(f_k)) \vee \\
(x \in \upsilon_n(f_k) \wedge y \in \upsilon_n(e_k)) \vee \\
(x \in \upsilon_n(e_k) \wedge y \in \upsilon_n(e_k)) \vee \\
(x \in \upsilon_n(f_k) \wedge y \in \upsilon_n(f_k))) \\
\iff \\
(\forall_{\neg C(e_k, f_k) \in \beta})\neg((\eta(e_k, x) = 1 \wedge \eta(f_k, y) = 1) \vee \\
(\eta(f_k, x) = 1 \wedge \eta(e_k, y) = 1) \vee \\
(\eta(e_k, x) = 1 \wedge \eta(e_k, y) = 1) \vee \\
(\eta(f_k, x) = 1 \wedge \eta(f_k, y) = 1))
\end{aligned}$$

Deffinition 36. *Let $R^v \subseteq W^{v2}$ be the set of all valid connected pairs of β modal points.*

$$R^v = \{< x, y \ \mid x, y \in W^v \wedge < x, y > \ is\ a\ valid\ connected\ pair\ of\ \beta\ modal\ points\} \tag{17}$$

Lemma 37. *Let $\mathcal{F} = (W^v, R^v)$ be a relational system. $\mathcal{F}$ satisfies the contacts and non-zero terms in β iff:*

$$\begin{aligned}(\forall(d_j \neq 0) \in \beta)(\exists \mathcal{E}_n \in W^v)(\mathcal{E}_n \in \upsilon_n(d_j)) \wedge \\ (\forall_{C(a_i,b_i)\in\beta})(\exists < x, y > \in R^v)(x \in \upsilon_n(a_i) \wedge y \in \upsilon_n(b_i)) \\ \Longleftrightarrow \\ (\forall(d_j \neq 0) \in \beta)(\exists \mathcal{E}_n \in W^v)(\eta(d_j, \mathcal{E}_n) = 1 \wedge \\ (\forall_{C(a_i,b_i)\in\beta})(\exists < x, y > \in R^v)(\eta(a_i, x) = 1 \wedge \eta(b_i, y) = 1)\end{aligned}$$

Let $\mathcal{F} = (W^v, R^v)$. Let $\mathcal{M} = (\mathcal{F}, \upsilon_n)$. $\mathcal{M}$ satisfies the zero terms and non-contacts in β. $\mathcal{M}$ is a model of β if the non-zero terms and contacts are satisfied 37 .

The model $\mathcal{M}$ is not convenient. It might have a lot of modal points and connections between them. For example, let $\beta = C((x \sqcap y) \sqcap z, t \sqcup e)$. Then $\mathcal{V}_n = \{x, y, z, t, e\}$. There are no zero terms and non-contacts, so W^v is the same as W_n. Therefore, $|W^v| = 32$ and $|R^v| = 32^2$.

The following algorithm builds a more convenient model. Creates modal points and connections only for the non-zero and contact terms in β.

```
1  $W \leftarrow \emptyset$
2  $R \leftarrow \emptyset$
   /* Process the non-zero terms in $\beta$ */
3  for $d_j \neq 0 \in \beta$ do
4      for $\mathcal{E}_n \in W^v$ do
5          if $\eta(d_j, \mathcal{E}_n) = 1$ then
6              $W \leftarrow W \cup \{x\}$
7              $R \leftarrow R \cup \{< x, x >\}$
8              go to 3
9      end
10     Unable to construct a model.
11 end
   /* Process the contacts in $\beta$ */
12 for $C(a_i, b_i) \in \beta$ do
13     for $< x, y > \in R^v$ do
14         if $\eta(a_i, x) = 1 \wedge \eta(b_i, x) = 1$ then
15             $W \leftarrow W \cup \{x, y\}$
16             $R \leftarrow R \cup \{< x, x >, < y, y >, < x, y >, < y, x >\}$
17             go to 12
18     end
19     Unable to construct a model.
20 end
21 Successfully constructed a model $\mathcal{M} = ((W, R), \upsilon_n)$.
```

3 Connected Contact Logics

3.1 Connectivity

In topology and related branches of mathematics, a connected space is a topological space that cannot be represented as the union of two or more disjoint non-empty open subsets. Connectedness is one of the principal topological properties that are used to distinguish topological spaces.

Axiom 38. *(Connectivity) Let b is a term. Then*

$$b \neq 0 \wedge b \neq 1 \implies bCb^* \tag{18}$$

Let $\mathcal{F}$ = (W, R) be a relational system with W ≠ $\emptyset$, R $\subseteq W^2$ and a, b are terms. Let us recall the deficition of C:

$$aCb \iff (\exists x \in \upsilon(a))(\exists y \in \upsilon(b))(xRy) \tag{19}$$

The connectivity theorem can be written as follows:

$$\upsilon(b) \neq \emptyset \wedge \upsilon(b) \neq W \implies (\exists x \in \upsilon(b))(\exists y \in W \setminus \upsilon(b))(xRy) \tag{20}$$

Deffinition 39. *Let G = (W, R) be a graph. W is the set of vertexes and R the set of edges. A **path** $\pi(v_1, v_n)$ is a sequence of vertexes $(v_1, v_2, \dots, v_n)$ such that $v_1, \dots, v_n \in V$ and $v_i R v_{i+1}$, for $i \in \{1, \dots, n-1\}$.*

Deffinition 40. *Let $G = (W, R)$ be an undirected graph. W is the set of vertexes and R the set of edges. G is* **connected** *if there is a path between each two vertices in W.*

$$(\forall x \in W)(\forall y \in W)(\exists \pi(x, y))(\pi(x, y) \ is\ a\ path\ in\ G) \tag{21}$$

Let $G = (W, R)$. The connectivity theorem implies that there is a connection out of each evaluation set(except the $\emptyset$ and W). Mind that the modal point representation is a variable evaluation. Thus, the set of all evaluations is $\mathcal{P}(W)$.

If G is connected, then there is a path between each two points. Therefore, there is a connection out of each $W' \in \mathcal{P}(W)$.

$$(\forall W' \in \mathcal{P}(W))(W' \neq \emptyset \wedge W' \neq W \implies (\exists x \in W')(\exists y \in W \setminus W)(xRy)) \tag{22}$$

If G is disconnected, then G has a not connected components. Let G'=(W', R') be such a component. Thus, $W' \subseteq W$ and $W' \in \mathcal{P}(W)$. There are no outgoing connections from W'. Otherwise, G' would be connected with at least one more point from $W \setminus W'$.

$$(\exists W' \in \mathcal{P}(W))(W' \neq \emptyset \wedge W' \neq W \wedge (\nexists y \in W \setminus W')(xRy)) \tag{23}$$

Therefore, the connectivity theorem implies that the graph G is connected.

Deffinition 41. *Let $\mathcal{F}$ = (W, R) be a relational system. Let $\mathcal{M} = (\mathcal{F}, \upsilon_n)$ be a model of β. $\mathcal{M}$ is a* ***connected model*** *if $G = (W, R)$ is connected.*

3.2 Connected model building

Let β be a branch conjunction as in 16 :

$$\bigwedge_{i=1}^{I} C(a_i, b_i) \ \wedge \ \bigwedge_{j=1}^{J} d_j \neq 0 \ \wedge \ \bigwedge_{k=1}^{K} \neg C(e_k, f_k) \ \wedge \ \bigwedge_{l=1}^{L} g_l = 0$$

Let $\mathcal{F}^v = (W^v, R^v)$ be a relational system. Let $\mathcal{M}^v = (\mathcal{F}^v, \upsilon_n)$ be a model. $\mathcal{M}^v$ is a model of β if the non-zero terms and contacts in β are satisfied 37 . If $\mathcal{M}^v$ is not a model of β, then β does not have a model neither a connected model.

Let $\mathcal{M}^v$ is a model of β. $\mathcal{M}^v$ is the biggest model of β w.r.t unique modal points and connections between them. Let $G^v = (W^v, R^v)$ be the graph of $\mathcal{F}^v$.

Deffinition 42. *Let G'=(W', R') and G"=(W", R") are an undirected graphs. G' and G'* **does not overlap** *iff:*

$$W' \cap W'' = \emptyset \wedge R' \cap R'' = \emptyset \tag{23}$$

Deffinition 43. *Let $F = (W, R)$ be a relational system. Let $G = (W, R)$ be an undirected graph. Then $Comp^G$ is the splitting of G to it's* **connected components** .

$$\begin{gathered} Comp^G = \{G' = (W', R') \mid W' \subseteq W \wedge R' \subseteq R \wedge G' \textit{ is connected}\} \wedge \\ (\forall\ G' \in Comp^G)(\forall\ G'' \in Comp^G)(G' \textit{ and } G'' \textit{ does not overlap}) \wedge \\ \{W' \mid G' = (W', R') \in Comp^G\} = W \wedge \{R' \mid G' = (W', R') \in Comp^G\} = R \end{gathered}$$

Let $Comp^{G^v}$ is the set of the connected components of G^v. If $Comp^{G^v}$ contains a graph which defines a model of β, then it is a connected model, too. Otherwise, β does not have a connected model.

Let $G' \in Comp^{G^v}$. Let $\mathcal{M}' = (W', R', \upsilon_n)$ is a model. In order $\mathcal{M}'$ to be a model of β the non-zero terms and contacts in β should be satisfied 37 . The zero terms and non-contacts are satisfied because $\mathcal{M}'$ has not introduces new modal points neither a connections.

4 Implementation Introduction

The main programming language is C++. Flex & Bison are used to parse the input formula. The formula proover is a C++ library. The unit and performance tests are C++ applications. The user application is a Web page. The Web server is implemented with the third party CppRestSDK library. The satisfiability checking runs on the server. There is a feature to interupt an ongoing process. Can be triggered by the user via a button. There is a user disconnecting detection which cancels the requested formula prooving. TODO: something more? maybe some screenshots from the web application or in it's section is enough?

4.1 Syntax

The formula should be easy and intuitive to write. Only the keyboard keys should be used. The legend bellow describes the formula's syntaxis:

Terms		
0	0	Boolean constant 0
1	1	Boolean constant 1
-	$*$	Boolean complement
*	$\cap$	Boolean meat
+	$\cup$	Boolean join
()	()	Parentheses
[a-zA-Z0-9]+	x_1	Boolean variable. Syntax $x1$, $Y42$, $Var101$

Formulas		
F	$\perp$	Propositional constant false
T	$\top$	Propositional constant true
~	$\neg$	Negation
&	$\wedge$	Conjunction
|	$\vee$	Disjunction
->	$\Rightarrow$	Implication
<->	$\Leftrightarrow$	Equivalence
C	C	Contact, syntax C(t_1, t_2)
<=	$\leq$	Part of , syntax <=(t_1, t_2)
<=m	$\leq m$	Measured Part of, syntax <=m(t_1, t_2)
=0	=0	Zero term, syntax $t_1 = 0$
()	()	Parentheses

These are a few examples of formulas:

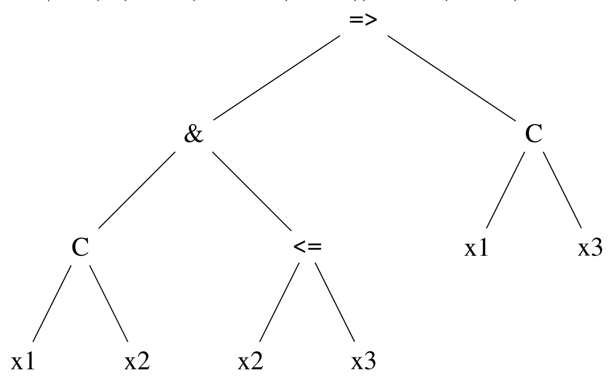
- $C(x1 * 1, x2 + y1)$
- $C(x1 + 0, (-x2 + x3) * x1)$
- $C(x1, x2)$ & $C(x2, x3)$ & $\sim C(x1, x3)$
- $C(x1, x2)$ & $<= (x1, x3)$ & $\sim C(x2, x3)$

- $C(x1, x2)$ -> $C(x2, x1)$
- $C(x1, x2)$ & $C(x2, x3)$ => $C(x1, x3)$
- F -> $C(x1, x2)$ & ~ $C(x1, x2)$

4.2 Formula parsing

The formula is a sequence of characters. These characters does not give us any information for the formula's structure. It should be analyzed. Flex [**?**] and Bison [**?**] are used to parse it into an AST (Abstract Syntax Tree). Flex is used as a tokenizer. Bison is used as the parser. The AST holds the formula's operations in the inner nodes. The terms are in the leaves.

Let ϕ = ($C(x1, x2)$ & <= $(x2, x3)$) => $C(x1, x3)$. The following is an AST of ϕ:



4.2.1 Tokenizer

The tokenizer is responsible for demarcating the special sybmols in the input formula. After the symbols are identified a token is created for each of them or at least for those significant to the semantic of the input formula. For example, the whitespaces are not significant and a tokens are not created for them. We shall use Flex as a tokenizer [**?**].

Grammar The tokenizer's grammar is composed from two types of tokens. Single character and multi character.

The Single character tokens are directly matched in the input formula and are representing the token itself. The multi character token is a sequence of characters which have some meaning when bundled together. This tokenizer's grammar is unambiguous and each input formula is uniquely tokenized.

The tokens derivation is explaned in details in the following table with Flex syntax. The matched symbol represents the symbol from the input formula and the output token is the newly created token for the matched symbol.

Matched sequence	**Output token**
[\t\n]	;
[,TF01()C&| *+-]	yytext[0];
"<="	T_LESS_EQ;
"<=m"	T_MEASURED_LESS_EQ;
"= 0"	T_EQ_ZERO;
"->"	T_FORMULA_OP_IMPLICATION;
"<->"	T_FORMULA_OP_EQUALITY;
[a-zA-Z0-9]+	T_STRING;
.	yytext[0];

Let us review the table above. All white spaces, tabulations and newlines are ignored. The syntax for it is the **;** character.

All single character tokens are passed as their ASCII code. The syntax for it is **yytext[0]**. It gives the matched character. That way it will be easy to use them in the parser.

The multi character tokens are converted to unique identificators. For example, the "<=" sequence is converted to T_LESS_EQ. The sequence of letters and numbers is converted to T_STRING. Later, it will be used as term's variables.

The last matched symbol in the table represents everything else, if nothing has been matched then just return the text itself. The parser will use it to promt where the unrecongized symbol was found and the symbol itself can be printed out.

4.2.2 Parser

The single character tokens are passed as their ASCII symbol to Bison. As discussed above the multi character tokens need more clearance in order to represent the literal from the input text symbols. The followings are definition of literals for multi character tokens:

- %token <const char*> T_STRING is the literal for "string"
- %token T_LESS_EQ is the literal for "<="
- %token T_MEASURED_LESS_EQ is the literal for "<=m"
- %token T_EQ_ZERO is the literal for "=0"
- %token T_FORMULA_OP_IMPLICATION is the literal for "->"
- %token T_FORMULA_OP_EQUALITY is the literal for "<->"

The followings are definitions of priority and associativity of the operation tokens. The priority is from low to hight (w.r.t. the line order in which they are defined)

- %left T_FORMULA_OP_IMPLICATION T_FORMULA_OP_EQUALITY
- %left '|' '+'

- %left '&' '*'
- %right '~' '-'
- %nonassoc '(' ')'

Parser grammar With the usage of the Parser literals, the input formula can be parsed to an Abstract Syntax Tree(AST). The AST contains all the data from the input string formula in a more structed way. On the AST additional optimizations can be done which will simplify the initial formula. It will produce better performance when a model is seeked in the satisfiability algorithms.

For convenience, will define two helper methods. Namely, **create_term_node** and **create_formula_node**. Both method construct AST nodes.

The create_term_node method creates an AST term node. It's arguments are an operation and up to two child terms. Depending on the operation arity.

The create_formula_node is analogious to the create_term_node method. Creates an AST formula node.

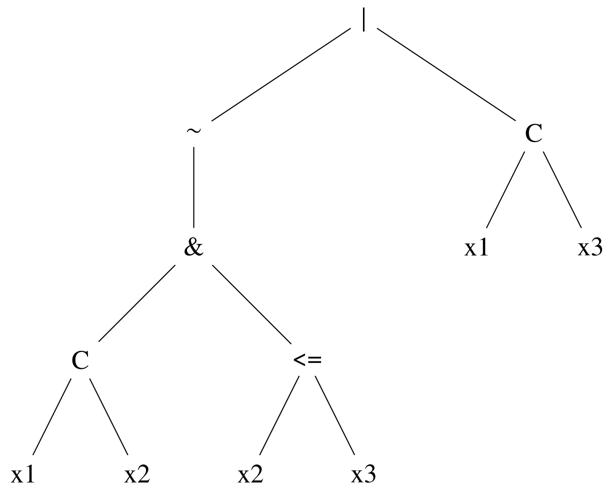
Few special symbols to define beforehand:

- **$$** is the return value to the 'parent'. Later, he can use it, e.g. as a child.
- **$i** is the return value of the i-th matched element in the matcher sequence.

Parser Algorithm The following is the parser algorithm which produces an Abstract Sytax Tree.

```
formula // 'formula' non-terminal
    : 'T' { // matching token 'T'
        $$ = create_formula_node(constant_true);
    }
    | 'F' {
        $$ = create_formula_node(constant_false);
    }
    | 'C' '(' term ',' term ')' {
        $$ = create_formula_node(contact, $3, $5);
    }
    | "<=" '(' term ',' term ')' {
        $$ = create_formula_node(less_eq, $3, $5);
    }
    | "<=m" '(' term ',' term ')' {
        $$ = create_formula_node(measured_less_eq, $3, $5);
    }
    | term "=0" {
        $$ = create_formula_node(eq_zero, $1);
    }
    | '(' formula '&' formula ')' {
        $$ = create_formula_node(conjunction, $2, $4);
    }
    | formula '&' formula {
        $$ = create_formula_node(conjunction, $1, $3);
    }
    | '(' formula '|' formula ')' {
        $$ = create_formula_node(disjunction, $2, $4);
    }
    | formula '|' formula {
        $$ = create_formula_node(disjunction, $1, $3);
    }
    | '~' formula {
        $$ = create_formula_node(negation, $2);
    }
    | '(' formula "->" formula ')' {
        $$ = create_formula_node(implication, $2, $4);
    }
    | formula "->" formula {
        $$ = create_formula_node(implication, $1, $3);
    }
    | '(' formula "<->" formula ')' {
        $$ = create_formula_node(equality, $2, $4);
    }
    | formula "<->" formula {
        $$ = create_formula_node(equality, $1, $3);
    }
    | '(' formula ')' {
        $$ = $2;
    }
  ;
term
    : '1' {
        $$ = create_term_node(constant_true);
    }
    | '0' {
        $$ = create_term_node(constant_false);
    }
    | "string" {
        $$ = create_term_node(term_operation_t::variable);
        $$->variable = std::move(*$1);
        // the string is allocated from the
        // tokenizer and we need to free it
        free_lexer_string($1);
    }
    | '(' term '*' term ')' {
        $$ = create_term_node(intersection, $2, $4);
    }
    | term '*' term {
        $$ = create_term_node(intersection, $1, $3);
    }
    | '(' term '+' term ')' {
        $$ = create_term_node(union_, $2, $4);
    }
    | term '+' term {
        $$ = create_term_node(union_, $1, $3);
    }
    | '-' term {
        $$ = create_term_node(complement, $2);
    }
    | '(' term ')' {
        $$ = $2;
    }
  ;
```

The AST can be easily modified. One of which is removing the implications and equivalences. They are replaced by conjunction, disjunction and negation. This is convenient because it simplifies the tableau method. It does not have to handle implication and equivalence. The following is a modified AST of ϕ without the implication:



THE FOLLOWING ARE NOTES WHICH WILL BE REMOVED

Contents: (notes) Parsing: - Formula legend - Variables - Definition, what can be variables. (sequence of characters, etc..) - Flex & Bison for parsing. - AST building from it. - Reducing some constants, using the AST and virtual dispatching.

Converstion from AST to more suitable formula representation: - Variables - OÑĆÑŁĐűĐťĐţÑĄÑĆĐšÑŔĐšĐřĐijĐţ ÑŔ ÑĄ Đ£Đ¿ÑĂĐţĐťĐ¡ĐÿÑŔ Đ¡Đ¿ĐijĐţÑĂ Đš ÑĄĐ£ĐÿÑĄÑŁĐžĐř ÑĄ Đ£ÑĂĐ¿ĐijĐţĐ¡ĐżĐÿĐšĐÿ(i.e. instead of the variables their corresponding number) The variable names does not change durring the formula satisfiability checking, so their corresponding "idexes " could be used... - Hashes..(for equality) - The boolean variables are strings.(variables - ids), equality - hashes (they are details) - Term representation. - Formula representation.

- Tableau - the official doc is fairly good, use it

- variable evaluations - representation - plus one operation (Next) - marked bits only for the used variables - plus one on marked bits - contact matrix representation

5 Tableaux Implementation

6 Model Implementation

7 Connected Contact Logics Implementation

TODOs: - inductive def. for structures, e.g. terms - recursive for functions - R is reflexive and symetric - somewhere to mention it.

TODO: separate line for the authors

References

[1] Handbook of Tableau Methods M. DâĂŹAgostino, D. M. Gabbay, R. HÂĺahnle and J. Posegga, eds.

[2] Modal Logics for Region-based Theories of Space Philippe Balbiani, Thinko Tinchev, Dimiter Vakarelov

[3] Connected Space or go to the next url: `https://en.wikipedia.org/wiki/Connected_space`