Propositional Tableaus
Blackburn & Bos - Chapter 4

Nomi Meixner

May 4, 2010

Outline

Some Reminders
- The Querying Task
- The Consistency Checking Task
- The Informativity Checking Task
- Consistency ↔ Informativity

Propositional Tableaus
- Introduction
- Propositional Tableaus
- Examples
- Definition

Prolog Implementation
- Neglected Clauses

Conclusion

3 Inference Tasks - The Querying Task

- **Querying Task:**
  Given a model M and a first-order formula ϕ, is ϕ satisfied in M or not?
- $\rightarrow$ model checker

3 Inference Tasks - The Consistency Checking Task

- **Consistency Checking Task:**
  Given a first-order formula ϕ, is ϕ consistent (i.e. satisfiable) or inconsistent (i.e. unsatisfiable)?
- Example for inconsistency: Mia is happy. Mia is not happy.
- $\rightarrow$ search problem for (possibly infinite) models
  - $\forall x \exists y \textsc{biggerthan}(x, y) \wedge \forall x \neg \textsc{biggerthan}(x, x) \wedge \forall x \forall y \forall z (\textsc{biggerthan}(x, y) \wedge \textsc{biggerthan}(y, z) \rightarrow \textsc{biggerthan}(x, z))$.
- $\rightarrow$ computationally undecidable task

3 Inference Tasks - The Informativity Checking Task

- **Validity:**
  A valid formula is a formula that is satisfied in all models (of the appropriate vocabulary) given any variable assignment. $\rightarrow$ if ϕ is a valid formula, it is impossible to find a situation and a context in which ϕ is not satisfied.
- **Informativity Checking Task:**
  Given a first-order formula ϕ, is ϕ informative (i.e. invalid) or uninformative (i.e. valid)?
- Example for uninformativity: $\textsc{robber}(x) \vee \neg\textsc{robber}(x)$

Consistency ↔ Informativity

- ϕ consistent iff $\neg\phi$ informative
  - ϕ consistent → satisfiable in at least one model → at least one model where $\neg\phi$ is not satisfied → $\neg\phi$ informative
- ϕ inconsistent iff $\neg\phi$ uninformative
  - ϕ inconsistent → not satisfiable in any model → $\neg\phi$ satisfied in every model → $\neg\phi$ uninformative
- ϕ informative iff $\neg\phi$ consistent

Consistency ↔ Informativity

- ϕ consistent iff $\neg\phi$ informative
  - ϕ consistent $\rightarrow$ satisfiable in at least one model $\rightarrow$ at least one model where $\neg\phi$ is not satisfied $\rightarrow$ $\neg\phi$ informative
- ϕ inconsistent iff $\neg\phi$ uninformative
  - ϕ inconsistent $\rightarrow$ not satisfiable in any model $\rightarrow$ $\neg\phi$ satisfied in every model $\rightarrow$ $\neg\phi$ uninformative
- ϕ informative iff $\neg\phi$ consistent
  - ϕ informative $\rightarrow$ at least one model where ϕ is not satisfied $\rightarrow$ at least one model where $\neg\phi$ is satisfied $\rightarrow$ $\neg\phi$ consistent

Consistency $\leftrightarrow$ Informativity

- ϕ consistent iff $\neg\phi$ informative
  - ϕ consistent $\rightarrow$ satisfiable in at least one model $\rightarrow$ at least one model where $\neg\phi$ is not satisfied $\rightarrow$ $\neg\phi$ informative
- ϕ inconsistent iff $\neg\phi$ uninformative
  - ϕ inconsistent $\rightarrow$ not satisfiable in any model $\rightarrow$ $\neg\phi$ satisfied in every model $\rightarrow$ $\neg\phi$ uninformative
- ϕ informative iff $\neg\phi$ consistent
  - ϕ informative $\rightarrow$ at least one model where ϕ is not satisfied $\rightarrow$ at least one model where $\neg\phi$ is satisfied $\rightarrow$ $\neg\phi$ consistent
- ϕ uninformative iff $\neg\phi$ inconsistent

Consistency $\leftrightarrow$ Informativity

- ϕ consistent iff $\neg\phi$ informative
  - ϕ consistent $\rightarrow$ satisfiable in at least one model $\rightarrow$ at least one model where $\neg\phi$ is not satisfied $\rightarrow$ $\neg\phi$ informative
- ϕ inconsistent iff $\neg\phi$ uninformative
  - ϕ inconsistent $\rightarrow$ not satisfiable in any model $\rightarrow$ $\neg\phi$ satisfied in every model $\rightarrow$ $\neg\phi$ uninformative
- ϕ informative iff $\neg\phi$ consistent
  - ϕ informative $\rightarrow$ at least one model where ϕ is not satisfied $\rightarrow$ at least one model where $\neg\phi$ is satisfied $\rightarrow$ $\neg\phi$ consistent
- ϕ uninformative iff $\neg\phi$ inconsistent
  - ϕ uninformative $\rightarrow$ satisfied in every model $\rightarrow$ $\neg\phi$ not satisfied in any model $\rightarrow$ $\neg\phi$ inconsistent

Chapter 4

- theorem proving methods for the quantifier-free fragment of first order logic (= propositional logic) $\rightarrow$ computationally decidable
- partial solution to consistency and informativity checking tasks
- consistency and informativity checking are actually closely related tasks, as both can be defined in terms of validity:
  - ψ is **un**informative wrt $\phi_1, ..., \phi_n$:
    $\phi_1, ..., \phi_n \models \psi$
    means (by Semantic Deduction Theorem):
    $\phi_1 \wedge ... \wedge \phi_n \rightarrow \psi$ is valid
  - ψ is **in**consistent wrt $\phi_1, ..., \phi_n$:
    $\phi_1, ..., \phi_n \models \neg\psi$
    means (by Semantic Deduction Theorem):
    $\phi_1 \wedge ... \wedge \phi_n \rightarrow \neg\psi$ is valid
- Needed: Computational technique to determine validity of a formula

Theorem Provers

- Validity is semantic concept
- Theorem Provers only work on symbols → no generation of (esp. infinite) models involved
- well-known and wide-spread: Tableau and Resolution

Advantages of Tableaus

- good for paper-and-pencil calculation
- mirrors semantic intuitions → *Semantic Tableaus*
- suitable for implementation

Propositional Tableaus - Intuition

- Given a formula and one of the truth values TRUE or FALSE is it possible to find a model in which the given formula has the given truth value?
- valid means true in all models → formula is valid if it is not possible to falsify it in any model
- Tableaus are means to show that all attempts to falsify a formula must fail → also called *refutation proof method*
- Tableaus use propositional logic without inner structure of atomic symbols, but uses sentence symbols instead:
  - $(\textsc{playsAirGuitar}(\textsc{vincent}) \rightarrow \textsc{happy}(\textsc{mia})) \wedge (\neg\textsc{playsAirGuitar}(\textsc{vincent}) \rightarrow \textsc{happy}(\textsc{yolanda}))$ simply becomes: $(p \rightarrow q) \wedge (\neg p \rightarrow r)$

Examples

Definition

A propositional formula ϕ is tableau-provable if and only if it is possible to expand the initial tableau consisting of the single node Fϕ to a closed tableau. We use the notation $\vdash_t \phi$ to indicate that ϕ is tableau-provable. If a formula is tableau-provable, then we say it is a tableau-theorem, or more simply, a theorem.

Prolog Implementation

```
tprove(F):-
   (
    closedTableau([[f(F)]]), !,
    write('Theorem.'), nl
   ;
    write('Not a theorem.'), nl
   ).
```

- F: imp(p,and(p,or(not(q),p)))

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[f(imp(p,and(p,or(not(q),p))))]]
- TempTableau:
- NewTableau:

Prolog Implementation - Contd.

```
expand([Branch|Tableau],[NewBranch|Tableau]):-
    unaryExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch|Tableau]):-
    conjunctiveExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch1,NewBranch2|Tableau]):-
    disjunctiveExpansion(Branch,NewBranch1,NewBranch2), !.

expand([Branch|Rest],[Branch|Newrest]):-
    expand(Rest,Newrest).
```

- Branch: [f(imp(p,and(p,or(not(q),p))))]
- Tableau: []
- NewBranch:

Prolog Implementation - Contd.

```
conjunctiveExpansion(Branch,[Comp1,Comp2|Temp])
    :-
     conjunctive(SignedFormula,Comp1,Comp2),
     removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [f(imp(p,and(p,or(not(q),p))))]
- Comp1:
- Comp2:
- SignedFormula:
- Temp:

Prolog Implementation - Contd.

```
conjunctive(t(and(X,Y)),t(X),t(Y)).
conjunctive(f(or(X,Y)),f(X),f(Y)).
conjunctive(f(imp(X,Y)),t(X),f(Y)).
```

- X:
- Y:

Prolog Implementation - Contd.

```
conjunctiveExpansion(Branch,[Comp1,Comp2|Temp])
    :-
    conjunctive(SignedFormula,Comp1,Comp2),
    removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [f(imp(p,and(p,or(not(q),p))))]
- Comp1: t(X)
- Comp2: f(Y)
- SignedFormula: f(imp(X,Y))
- Temp:
- X:
- Y:

Prolog Implementation - Contd.

```
conjunctiveExpansion(Branch,[Comp1,Comp2|Temp])
    :-
    conjunctive(SignedFormula,Comp1,Comp2),
    removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [f(imp(p,and(p,or(not(q),p))))]
- Comp1: t(p)
- Comp2: f(and(p,or(not(q),p)))
- SignedFormula: f(imp(X,Y))
- Temp: []
- X: p
- Y: and(p,or(not(q),p))

Prolog Implementation - Contd.

```
expand([Branch|Tableau],[NewBranch|Tableau]) :-
    unaryExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch|Tableau]) :-
    conjunctiveExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch1,NewBranch2|
     Tableau]) :-
    disjunctiveExpansion(Branch,NewBranch1,
        NewBranch2), !.

expand([Branch|Rest],[Branch|Newrest]) :-
    expand(Rest,Newrest).
```

- Branch: [f(imp(p,and(p,or(not(q),p))))]
- Tableau: []
- NewBranch: [t(p),f(and(p,or(not(q),p)))]

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[f(imp(p,and(p,or(not(q),p))))]]
- TempTableau: [[t(p),f(and(p,or(not(q),p)))]]
- NewTableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|
    Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [t(p),f(and(p,or(not(q),p)))]
- Rest: []
- Tableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|
     Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [t(p),f(and(p,or(not(q),p)))]
- Rest: []
- Tableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [t(p),f(and(p,or(not(q),p)))]
- Rest: []
- Tableau: []

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[f(imp(p,and(p,or(not(q),p))))]]
- TempTableau: [[t(p),f(and(p,or(not(q),p)))]]
- NewTableau: [[t(p),f(and(p,or(not(q),p)))]]

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[t(p),f(and(p,or(not(q),p)))]]
- TempTableau:
- NewTableau:

Prolog Implementation - Contd.

```
expand([Branch|Tableau],[NewBranch|Tableau]):-
    unaryExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch|Tableau]):-
    conjunctiveExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch1,NewBranch2|
    Tableau]):-
    disjunctiveExpansion(Branch,NewBranch1,
        NewBranch2), !.

expand([Branch|Rest],[Branch|Newrest]):-
    expand(Rest,Newrest).
```

- Branch: [t(p),f(and(p,or(not(q),p)))]
- Tableau: []
- NewBranch1:
- NewBranch2:

Prolog Implementation - Contd.

```
disjunctiveExpansion(Branch,[Comp1|Temp],[Comp2|
      Temp]):-
    disjunctive(SignedFormula,Comp1,Comp2),
    removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [t(p),f(and(p,or(not(q),p)))]
- Comp1:
- Comp2:
- SignedFormula:
- Temp:

Prolog Implementation - Contd.

```
disjunctive(f(and(X,Y)),f(X),f(Y)).
disjunctive(t(or(X,Y)),t(X),t(Y)).
disjunctive(t(imp(X,Y)),f(X),t(Y)).
```

- X:
- Y:

Prolog Implementation - Contd.

```
disjunctiveExpansion(Branch,[Comp1|Temp],[Comp2|
    Temp]):-
      disjunctive(SignedFormula,Comp1,Comp2),
      removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [t(p),f(and(p,or(not(q),p)))]
- Comp1: f(X)
- Comp2: f(Y)
- SignedFormula: f(and(X,Y))
- Temp:
- X:
- Y:

Prolog Implementation - Contd.

```
disjunctiveExpansion(Branch,[Comp1|Temp],[Comp2|
    Temp]):-
    disjunctive(SignedFormula,Comp1,Comp2),
    removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [t(p),f(and(p,or(not(q),p)))]
- Comp1: f(p)
- Comp2: f(or(not(q),p))
- SignedFormula: f(and(X,Y))
- Temp: t(p)
- X: p
- Y: or(not(q),p)

Prolog Implementation - Contd.

```
expand([Branch|Tableau],[NewBranch|Tableau]):-
    unaryExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch|Tableau]):-
    conjunctiveExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch1,NewBranch2|Tableau]):-
    disjunctiveExpansion(Branch,NewBranch1,NewBranch2), !.

expand([Branch|Rest],[Branch|Newrest]):-
    expand(Rest,Newrest).
```

- Branch: [t(p),f(and(p,or(not(q),p)))]
- Tableau: []
- NewBranch1:[f(p),t(p)]
- NewBranch2: [f(or(not(q),p)),t(p)]

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[t(p),f(and(p,or(not(q),p)))]]
- TempTableau: [[f(p),t(p)],[f(or(not(q),p)),t(p)]]
- NewTableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(p),t(p)]
- Rest: [f(or(not(q),p)),t(p)]
- Tableau:

Prolog Implementation - Contd.

```
closedBranch(Branch):-
        memberList(t(X),Branch),
        memberList(f(X),Branch).
```

- Branch: [f(p),t(p)]
- X:

Prolog Implementation - Contd.

```
closedBranch(Branch):-
        memberList(t(X),Branch),
        memberList(f(X),Branch).
```

- Branch: [f(p),t(p)]
- X: p

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(p),t(p)]
- Rest: [f(or(not(q),p)),t(p)]
- Tableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(or(not(q),p)),t(p)]
- Rest: []
- Tableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(or(not(q),p)),t(p)]
- Rest: []
- Tableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch),  !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|
    Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(or(not(q),p)),t(p)]
- Rest: []
- Tableau: []

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[t(p),f(and(p,or(not(q),p)))]]
- TempTableau: [[f(p),t(p)],[f(or(not(q),p)),t(p)]]
- NewTableau: [[f(or(not(q),p)),t(p)]]

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[f(or(not(q),p)),t(p)]]
- TempTableau:
- NewTableau:

Prolog Implementation - Contd.

```
expand([Branch|Tableau],[NewBranch|Tableau]):-
    unaryExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch|Tableau]):-
    conjunctiveExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch1,NewBranch2|
     Tableau]):-
    disjunctiveExpansion(Branch,NewBranch1,
        NewBranch2), !.

expand([Branch|Rest],[Branch|Newrest]):-
    expand(Rest,Newrest).
```

- Branch: [f(or(not(q),p)),t(p)]
- Tableau: []
- NewBranch:

Prolog Implementation - Contd.

```
conjunctiveExpansion(Branch,[Comp1,Comp2|Temp])
    :-
    conjunctive(SignedFormula,Comp1,Comp2),
    removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [f(or(not(q),p)),t(p)]
- Comp1:
- Comp2:
- SignedFormula:
- Temp:

Prolog Implementation - Contd.

```
conjunctive(t(and(X,Y)),t(X),t(Y)).
conjunctive(f(or(X,Y)),f(X),f(Y)).
conjunctive(f(imp(X,Y)),t(X),f(Y)).
```

- X:
- Y:

Prolog Implementation - Contd.

```
conjunctiveExpansion(Branch,[Comp1,Comp2|Temp])
    :-
    conjunctive(SignedFormula,Comp1,Comp2),
    removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [f(or(not(q),p)),t(p)]
- Comp1: f(X)
- Comp2: f(Y)
- SignedFormula: f(or(X,Y))
- Temp:
- X:
- Y:

Prolog Implementation - Contd.

```
conjunctiveExpansion(Branch,[Comp1,Comp2|Temp])
    :-
    conjunctive(SignedFormula,Comp1,Comp2),
    removeFirst(SignedFormula,Branch,Temp).
```

- Branch: [f(or(not(q),p)),t(p)]
- Comp1: f(not(q))
- Comp2: f(p)
- SignedFormula: f(or(not(q),p))
- Temp: t(p)
- X: not(q)
- Y: p

Prolog Implementation - Contd.

```
expand([Branch|Tableau],[NewBranch|Tableau]):-
    unaryExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch|Tableau]):-
    conjunctiveExpansion(Branch,NewBranch), !.

expand([Branch|Tableau],[NewBranch1,NewBranch2|
    Tableau]):-
    disjunctiveExpansion(Branch,NewBranch1,
        NewBranch2), !.

expand([Branch|Rest],[Branch|Newrest]):-
    expand(Rest,Newrest).
```

- Branch: [f(or(not(q),p)),t(p)]
- Tableau: []
- NewBranch: [f(not(q)),f(p),t(p)]

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[f(or(not(q),p)),t(p)]]
- TempTableau: [[f(not(q)),f(p),t(p)]]
- NewTableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|
    Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(not(q)),f(p),t(p)]
- Rest: []
- Tableau:

Prolog Implementation - Contd.

```
closedBranch(Branch):-
        memberList(t(X),Branch),
        memberList(f(X),Branch).
```

- Branch: [f(not(q)),f(p),t(p)]
- X:

Prolog Implementation - Contd.

```
closedBranch(Branch):-
        memberList(t(X),Branch),
        memberList(f(X),Branch).
```

- Branch: [f(not(q)),f(p),t(p)]
- X: p

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(not(q)),f(p),t(p)]
- Rest: []
- Tableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|
     Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(not(q)),f(p),t(p)]
- Rest: []
- Tableau:

Prolog Implementation - Contd.

```
removeClosedBranches([],[]).

removeClosedBranches([Branch|Rest],Tableau):-
    closedBranch(Branch), !,
    removeClosedBranches(Rest,Tableau).

removeClosedBranches([Branch|Rest],[Branch|
    Tableau]):-
    removeClosedBranches(Rest,Tableau).
```

- Branch: [f(not(q)),f(p),t(p)]
- Rest: []
- Tableau: []

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

- OldTableau: [[f(or(not(q),p)),t(p)]]
- TempTableau: [[f(not(q)),f(p),t(p)]]
- NewTableau: []

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
      expand(OldTableau,TempTableau),
      removeClosedBranches(TempTableau,NewTableau),
      closedTableau(NewTableau).
```

Prolog Implementation - Contd.

```
closedTableau([]).

closedTableau(OldTableau):-
    expand(OldTableau,TempTableau),
    removeClosedBranches(TempTableau,NewTableau),
    closedTableau(NewTableau).
```

Prolog Implementation

```
tprove(F):-
   (
    closedTableau([[f(F)]]), !,
    write('Theorem.'), nl
   ;
    write('Not a theorem.'), nl
   ).
```

- F: imp(p,and(p,or(not(q),p)))

Prolog Implementation - Neglected Clauses

```prolog
expand([Branch|Tableau],[NewBranch|Tableau]):-
     unaryExpansion(Branch,NewBranch),  !.
```

```prolog
expand([Branch|Rest],[Branch|Newrest]):-
     expand(Rest,Newrest).
```

```prolog
unaryExpansion(Branch,[Component|Temp]):-
     unary(SignedFormula,Component),
     removeFirst(SignedFormula,Branch,Temp).
```

```prolog
unary(t(not(X)),f(X)).
unary(f(not(X)),t(X)).
```

Conclusion

- paper-and-pencil version of Tableaus
- Prolog implementation of Tableaus

→ So far decidable, as only propositional logic covered

- Partial solution to the consistency and informativity checking tasks
- Determines validity of a formula
- Works on symbols

→ Exchangeable module in the big picture (cf. Resolution)