A Computational Approach to Classical Logics and Circuits

Cz = Reslock (C1,C4)=(2)*P*V (8) *P* and

*C*6 = Resock (C2,C3)=(4)9V6-*9* If we apply also the deletion strategy, and we delete these tautologies, then we cannot continue the derivation process to obtain a. Note that even if S is inconsistent, the empty clause (0) cannot be derived because there are too many restrictions imposed by lock resolution and the deletion strategy. Therefore*, lock resolution + the deletion strategy is not complete.*

Continuing the lock resolution process but using also the resolvents Cs and Co we obtain: C, = Res.ock *(*C2,C3)=(4*) 9V*18) *-* The clauses C, and C2 are similar but not identical, the literals order (provided by the indices) differs in these clauses: -*p* is the literal to resolve upon from C2, and *q* is the literal to resolve upon form C7. Thus, the role of a tautology as a parent clause is to modify the literals order in the other parent clause.

G=(2*)P*V(1*)9, C2*=(3) *7PV(*4)9, C3=(5) PV (6) -9, C4=18) *TPV* (7) *9* Cs=(2*) PV* (8) P, C6=(4*39V*6*-9 C7=(*4*)q*V(8) =P Cg = Reslock (C7, Ca)=(8) *P, C*o = Res\ck (C3,C3)=(6*) -9, C*10 = Resçock *(*C1,C9)=(2*)P,*

Cu = Resock (C10,C3)=0. As a conclusion, in this second indexing the empty clause was derived from S with the help of tautologies. **Exampl**e 5.9. In this example we shall prove that *lock resolution + the set-of-support strategy is not complete.* Using lock resolution prove the deduction:

*p+(9r), r*as*t , u* SAE *paq-U*.

The premises (hypotheses) of the deduction are:

*Uj = p(r*), *U2 =ras →t, U3 = u* →$^

The conclusion of the deduction is: *V =p^g-*>-*U*

129

Resolution Proof Method

The resolution is applied to the set of clauses S, obtained from the conjunctive normal forms of the hypotheses and the negation of the conclusion.

CNF*(U*1)=*pvzvr,*

Ci=*pvavr* CNF*(U*2)=7V-*VI,*

C2 ="V-*Vt* CNF(*U*3)=*(uv*s)^(-*uv*-t), Cz = n*uv*s, C4 = *UV-t* CNFGV)= *pagar,*

*Cg = P, Co=9, C, = u* S = {C1,*C2,C3,*C4,*C*5,*C6, Cy*} Using the set-of-support strategy we choose the support set Y = {C*5,C6,C*y} corresponding to the conclusion of the deduction.

S\Y = {C1,*C2*,*C*3,C4} is a consistent set of clauses corresponding to the premises (hypotheses) of the deduction. We index the literals from the clauses as follows:

C1=(3) *PV (2)* 79V (1)*”*, C2=(6) PV(5) -SV(4)ť, Cz=(8) *UV(7*)S, C4=(10) TUV(977,

*Cs*=(11)*P,* C*o*=(12*)9*, *C7*=(13)U Note that the restriction imposed by lock resolution (the literals of lowest indices in their clauses resolve) combined with the set-of-support strategy (avoids to resolve two clauses belonging to the consistent set S\Y = *{C*1,C2,C3,C4}) will block the resolution process and a cannot be derived. If we do not use the set-of-support strategy we can derive ..

Cg = Reslock (C2,C4)=(6) "V(5) -8V(10) T*u* Cg = Respock (C3,C8)=6)=rV(8) T*u C*vo = Reslock (C1,C9)=(3)PV(2) 9V(8) 7!!

,C10=(3) *P*V(8) C12 = Resock (C5,C1)=(8) T*u* C13 = Resock (C*,,*C12)=0

We conclude that the set S = {*C*1,*C*2,*C*3,C4,C5, *C6*,*C7*} is inconsistent and *the deduction p + (q + r*), *ra*s *→t, u* →SA- E *p^q+nu holds.*

130

A Computational Approach to Classical Logics and Circuits

The binary tree corresponding to the refutation process is depicted below.

C2=6=rV(5) -5V(4) C4=(10) TUV (9) \*\*

C 6) ="V(653V010 (3-9-u (7) C9=6) V(8) = C1=(3)–PV(2)-V012 610-3\*evoCo-on? CHF, PO C5-002

C12=18) T4 C1=(13)4

***5.4. Linear resolution***

Proposed by D.W. Loveland in 1970 [30], linear resolution is a very efficient refinement of resolution. The resolution process is a linear one: at each step one of the parent clauses is the resolvent derived at the previous step. For a set S of clauses, a *linear deduction o*f C, from S with Co es as the top clause is symbolized graphically as follows:

*C*o is the *top clause C1,*C2...C*n*-1*, C*, are *central clauses Bo, B1,..., By*-1 ar*e side clauses*

*Vi*= 1,.*..,N:C;* = Res(*Ci*-1*,B;*-1) *Vi=*0,...*,n-1:B;* ESU{Co*,C*...,Ci-2}

131

Resolution Proof Method

Linear resolution is sound and complete according to the theorem: **Theorem 5.9. Soundness and completenes**s [30] The set S of clauses is inconsistent if and only if s hes a.

We can combine the linear resolution with the deletion strategy and the completeness property is preserved. This refinement of resolution also provides a strategy at the implementation level: backtracking algorithm. In each iteration, for the current central clause there are more possible side clauses. We continue the resolution process choosing one side clause. If in the iteration *i* the process is blocked (the central clause C; is a tautology or it is an existing central clause: C; *=C,,j*<i) or all the side clauses of *C;* were used, then we go back to the previous iteration (*i*-1) and we choose another possible side clause for Ci-, to continue the resolution process.

The algorithm stops in two cases:

the empty clause was derived and the conclusion is that S is inconsistent.

• for the top clause all the possible side clauses were used, but the empty clause

was not derived, then we conclude that the set S is consistent. The consistency of a set of clauses is proved after a complete search without the derivation of the empty clause. Special cases of linear resolution [11]:

**• *unit resolut****ion:* the central clauses have at least a unit clause as a parent clause.

*• input resolution:* all side clauses are initial (input) clauses.

The equivalence of unit resolution and input resolution is expressed by the following theorem.

Theorem 5.10. [11] Let S be a set of clauses. S

est *o if and only if S* kes a.

These two refinements of resolution are sound, but they are not complete: 1. s**oundne**ss: If Shim*p*ul unil a then S is inconsistent; 2. **incompleteness**: there exist inconsistent sets of clauses from which the empty

clause cannot be derived using input or unit resolution.

132

A Computational Approach to Classical Logics and Circuits

Example 5.10. The set S =*{pvq, pvq,PV 79, PV*-9} is inconsistent (see Example 5.1), but there is no unit refutation from S because there is no unit clause in S. According to Theorem 5.10 there is no input refutation from S. This is an example of incompleteness of unit/input resolution. **Definition 5.3.** A clause is called a *positive clause* if it contains only positive literals. A clause is called a *negative clause* if it contains only negative literals. A clause is called *Horn clause* if it contains exactly one positive literals, all the other literals are negative.

**Theorem 5.11.** The input resolution is complete on a set of Horn clauses with a negative top clause. (PROLOG).

**Example 5.11. knowledge base** From the hypotheses:

*H:U, ^U2 ^... Un V H2:X*, ^ X2 1...1 X, →Y

*H,:*W, W2^.Wm *→R* is C = Z11 22 ...A Zm deducible? To prove by refutation using resolution we have to transform into clausal normal forms the hypotheses and the negation of the conclusion. A formula of type: *U,^U2 1... Un* →V provides a Horn clause:

*LUV-U2 V*..*. V-U*, V*V* The hypothesis *H;* provides the Horn clause C;,*i* = 1,2,..*.,j.* The negation of C is -(Z1^22 1... A Zm) and provides a negative clause:

*C*i+1=-ZV-Z2 V...V-2m We have to apply the input resolution to the set S={C1,C2,...*,C*,,Ci+1} with the top clause Cj+1. According to Theorem 5.11 we have that:

*H,H2,...,* FC if and only if SER

:f*a input*

Res

O.

**Example 5.12.** Using linear resolution prove the inconsistency of the sets of clauses:

S = {C1,C2,*C*3,C4,C5}.

133

Resolution Proof Method

*C = pvqvr, C2 = -2*, C3 = *7q*, C4 = -r Vw, C5 =-W The derivation tree is as follows:

Cs=5w C4 =-r vw Co== G = pv avr

G=pvq||C3=-9 C3= C2 = -2

Co=0

Top clause: Cs, side clauses: C4,C1*,C*3*,C*2, central clauses: C6*,C7,C*3*,Cg*. This linear resolution is also an *input* resolution and a *unit* resolution

Stikles , therefore S is an inconsistent set.

**Example 5.13.** Check the consistency/inconsistency of the set S of clauses using linear resolution.

S ={C1 = *pvq,C2* =*pvq,C*3 = *PV-9}* We begin the linear search of the derivation of the empty clause using the backtracking strategy: *First iteration:*

**Cis the top clause and it has 3 possible side clauses**: C2,*C*3,*C*3. Note that C and Cz can resolve in two ways: with *p* and q respectively as the literal resolved upon. I) C2 is **used as a side clause for** C.

*Second iteration:* C4 = Resp(C1,*C2*)*=q* - central clause having one possible side clause: C3

*Third iteration:* Cz = Res, *(C*4,C3)=*P* - central clause having one possible side clause: C *Fourth iteration:* Co = Resp(C5,C)=*q=C*4 (an existing clause) The search process is blocked, otherwise it will be an infinite loop.

134

A Computational Approach to Classical Logics and Circuits

We go back to the previous iteration *(third iteration)*: all the possible side clauses of Cs were used. We go back to the previous iteration (s*econd iteration*): all the possible side clauses of C4 were used. W*e* go back to the previous iteration *(first iteration).*

II) C3 **(with *p* as the literal resolved upon) is used as a side clause for C**

*Second iteration:* C7 = Res,(*C*,C3)=*9V-9=7* (tautology) The search process is blocked, because a tautology cannot help to derive o (inconsistency). We go back to the previous iteration (*first iteration).*

III) C3 (**with *9* as the literal resolved upon) is used as a side clause for a**

*C*g = Res,(C1,C3)*= pv p=T* (tautology) The search process is blocked. We go back to the previous iteration *(first iteration).*

All the possible side clauses of *C* were used. A complete linear derivation search was performed, but the empty clause was not derived, so S is a consistent set.

The graphical representation of the resolution process, following the backtracking strategy is depicted below.

1) C

C2,C3,C3

II

C

C2,C3,C3 III)

C2,C3,C3

C7 - tautology process blocked

Cg - tautology process blocked

C=C

process blocked

135

Resolution Proof Method

Ares

*5.5. Resolution in first-order (predicate) logic* We present the *predicate resolution a*s a formal (axiomatic) system according to the approach from the paper [63]: Res Pr = (Res, Fres, ARES*, RR*es), where:

• Res = pr-{-7, 47,1,5, V} is the alphabet;

*F*RIS V{o} is the set of well formed formulas; - *Fr*es is the set of all clauses built using the alphabet Eres;

is empty clause which does not contain any literal and symbolizes inconsistency;

= Ø is the set of axioms; *Rres = {res Pr, fact}* is the set of inference rules containing the *resolution rule (re*sPr) and the *factoring rule (fact).*

*f vh,gv-*22 Frespo 20*f*) v2(g), where a *= mgu(11,12), f,ge F*res

*4 Vla V... Vlk vlk+*1 V.*.. VI,* E *fact vlk*+*V...Vly*, where *= mgu(11,12,...,lk)* **Definit**ion 5.4. The predicate clauses *C = f vli* and C2 =*gv-12*, without common free variables, are called *clashing clauses* if the literals l, and *la* are unifiable: there exists 2 *= mgu(li,12).* Th*e binary resolvent* of G and C2 is C = Rest *(*C*,C*2) = 1(*f*) vag). If *C = l, vla V...Vlk Vlk + V...VIn* and a *= mgu(11,12,...,k),*

*Fact(C*)= 2*(1*) va*lk-*) v...*V* 2*(*is called a *factor* of *C*.

These two inference rules are combined and the definition of a resolvent is obtained **Definition** 5.5.

Th*e predicate resolvent* of two parent clauses C and C2 is one of the following: 1. the binary resolvent of *C* and C2; 2. the binary resolvent of C and a factor of C2; 3. the binary resolvent of a factor of *C* and C2; 4. the binary resolvent of a factor of C, and a factor of *C*2.

**Example 5.14.** a) *C = P(f(*x),g(y)*) VQ*(x, y) and C2 =*=P(f(f(a)),g(*z)) VQ*(f(a*),g(z))

Respir-f(a), ykz](C1,C2)=2*(f(a)*, g(z)*) v* Q*(f(a),z)*

136

A Computational Approach to Classical Logics and Circuits b) C = *P*(x,g(y),z) *v P(a,g(b), a) v* Q(x,z*)*

1*1 = P*(x,g(y),z*), l2 = P(a,g(6),a) Fact(C)*= M(C), where a =[x+a,

y *b*,z+a)*= mgu(11,12). Fact(C)=P(a,g(6),a) v Qla,a*) is a factor of C

**Theorem 5.12.** Let *U,,U2,...UM,V* be first-order formulas. 1. *V* is a theorem *if and only if (*n° FRes B.

**Pr**

*2. U1,U2*,.*..,U*, E*V if and only if {U',U2*,.*..,Un'*,(V)"}FRes o.

a

All the refinements and strategies for propositional logic can be used in predicate logic. It is recommended to rename the free variables in the initial set of clauses, such that they will be distinct in different clauses. *Algorithm pre****dicate\_resolution:*** input: *U,,U2,...U,V* - first-order formulas. output: message: ”*U,,U2,...U*nTV”or ,*U1,U2,...Un*HV” or

“we cannot decide if *U, U2,...U*, FV or *U1,U2....U, HV”* **begin**

build the set of clauses: S:=*{U*,^*,U2,...,U,^,(*-1)^}; **do{**

select *11,12,G*7,C2 such that:

*C*1,C2 are clauses or factors of clauses of S;

1, €C1, and -12 *€ C2*; if (I, and la are unifiable with *0 := mgu(11,12*)) **then**

C:= Rest *(*C*,*C2); if (C =o) then write “*U1, U2,...U*n V”; exit;

else S:=SU{C};

**end\_if end if** }until (no new resolvents can be derived or a predefined quantity of effort was

done) if (no new resolvents can be derived) then write ” *U1,U2,...U*n*HV”*;

else write "we cannot decide if *U*1*,U2....U*n TV or *U1,U2.... Un* HV” **end\_if end** The resolution algorithm for predicate logic is a semi-decision procedure.

137

Resolution Proof Method

**Theorem 5.13. *(*Church 1**936 (13]) The problem of validity of a first-order formula is *undecidable,* but it is *semi-decidable. I*f a procedure *Proc* is used to check the validity of a first-order formula *U* we have the following cases: 1. if the formula *U* is valid, then *Proc* ends with the corresponding answer. 2. if the formula *U* is not valid, then *Proc* ends with the corresponding answer or

*Proc* may never stop. Example 5.15. Check if *U*F*V*, where *U =* (Vx*)(P*(x) → *Q*(x)) and V = (x)*P*(x) → (Vx)Q(x).

We transform the formulas *U* and V into clausal normal forms:

• Prenex normal forms:

*UP* =(Vx)(~P(x) v Q(x*))* (V) =-*((Hz)P*(z)+(Wy)*Q*(y)) = *(*Hz*)P*(z) ^ (3y)-2(y)=

=(Hz)(3y)*(P*(x)^ 2(y)) Skolem normal forms: *US =* (Hx)(~P(x*) V*Q(x)),

(V)' = *(*V2)(P(z)^-*Q(f(z*))), *f* is a unary Skolem function

• Clausal normal forms: *UC =—P*(x) *v Q*(x),

(V) = *P(z)^-Q((z*)) We consider the following set of clauses: S =*{C*1,C2,*C*3}. The free variables are renamed such that they are distinct in the set S. *G =-P*(x) v Q(x), C2 = *P*(y), C3 =-*$(*z)) We derive the empty clause using predicate resolution as follows: C,C,EP C4 = 2(x) *C*4,C5 Flat S (3) According to Theorem 5.12 we proved that *U*FV.

**Example** 5.16. Using linear resolution prove that S = {C,C2,*C*3,C4} is an inconsistent set of clauses: *G = P(x, f*(x)*,e*), *C2=-R*(x) V-R(*y) V-P(x, f(*y),z) *v R(z), C*3 = *R(a*)*,*

*C*4 =-*R(e)*

Constants: *a,* e, function symbols: *f* , predicate symbols: *P,R*

138

20

*A* Computational Approach to Classical Logics and Circuits For linear resolution we choose the top clause: C4 *C*2 Haley ) *C*3 =-R(x)v*—P(x, f(*x),z*) v R*(z*), C*z is a factor of C2 *C*4,C5 HD Co=\_R(x) *v—P*(x*, f*(x),e), central clause *C*6, C3 ) Cy*=-P*(*a, f(a),* e), central clause C7, G FITA] Cg = 0 From the set S we derived the empty clause, therefore S *is inconsistent.* The refutation tree is depicted below.

reg Pr

©

©

top clause: C4

top clause: Ca

side clauses: C3,C3,

central clauses: *C6,C7,*C9

**Example** 5.17. Check the inconsistency of the set S of clauses using lock resolution. *S*={-*P*(x) *v* Q(x*) v R(*x), (y) *v R*(y), *P(a), -*R(*a*)}

a) We index the literals as follows:

*C*1=13) *P*(x)V(2) Q(x*)V*(1*) R*(x) C2=(5) -Q(y)V(4) R(y) C3=*16*P*(a),* C4=*(7) -R(a)* The following resolvents are obtained: Cs = Res[\*xa] (C1,C4)=(3) ~P(a)V(2)(a) Co = Res{y\_aj (C2,Ca)=(s)-0(a) C4 = Res”' (C5,Co)=(3)=*P(a) C*g = Resor (C6,*C*)=0 Shres and thus S *is inconsistent.*

139

Resolution Proof Method

b) another indexing of the literals:

C=(2)~P(x)V(1)Q(x) (3) R(x) C2=(4)O(y)*V(5) R(y) C3=(6P(a)* C4=17) -*R(a)* The following resolvents are derived: Cs = Restyr-x] (C1,C2)=(2) P(x)V(3*)R(*x) Co = Reserra] (C3*,*C5)=(3) R*(a)* C1 = Res'' (C4*,C*6)=0

Another lock refutation from S was obtained. Example 5.18. Prove the semidistributivity of ,,V” over „+”:

F(Vx)*(P(*x) → Q(x))((Vx*)P*(x) → (Vx)Q(x)) and

H((*Hx)*P(x) → (Vx)Q(x)) +(*V*x)*(P*(x) →Q(x)) by applying predicate resolution. We consider the predicate formulas.

*U; =*(Vx)*(P*(x) → Q(x))+((Vx*)P(*x) → (Vx*)Q*(x))

*U2 =((Vx)P*(x) =(Vx)Q(x))+(*V*x)(P(x) + Q(x)) According to Theorem 5.12:

• HU, if and only if (-*U*) Freso and

• *HU2 if and only if (-U*)CHRES O. We apply syntactic transformations in order to obtain the prenex, Skolem and clausal normal forms of the formulas -*U*, and -*U,*

*-U=-((V*x)(*P*(x) + Q(x))+*(*(Vx*)P*(x)+(Vx*)*Q(x)))

- the logical equivalence -(A → *B*) = AA*B* is applied

= ( x)*(P*(x) + Q(x))^ ((Vx*)P*(x) → (Vx)Q(x*)*) =

= (Vx)*(*P(x) →Q(x))^(Vx*)P(*x)^-(Vx)Q(x)

• infinitary DeMorgan's law: -(Vx)A(x) = (2x)A(x) is applied

=(Vx)(~P(x) *v*Q(x)*)*^(Vx*)P(*x)^(Ex)-Q(x) rename the bound variables =(Vx)(\_*P*(x*) V*Q*(*x))^(*Wy*)P(y*)*^(Ez)-&(z) extraction of the quantifiers in front of the formula, we extract first the existential quantifier = (z)(Vx)(Vy)((~P(x*) V*Q(x*)) ^ P*(y*) ^*-*Q*(z))=(-*0*.)P - prenex form

**ene**

140

A Computational Approach to Classical Logics and Circuits

*(U*n)=(Vx)(Wy)<(—P(x*)*v*Q(*x)*)^* P(y) ^\_Q(a)) - Skolem normal form

[

z *a*], *a*- Skolem constant *GU*) =*(~P*(x*) VQ*(x)*)^ P*(y)^-Q(a) - clausal normal form The set of clauses used in predicate resolution is:

Si = {C1 = *P*(x*)* v Q(x), C2 = *P*(y*), C*z = -Q(a)} The following resolvents are derived:

C4 = Resexraj*(*C*,C*3)=*P(a)*

Cg = Resfsaj(C2,Ca)=0 We proved: (-U) Fresa, thus +(4x)*(P*(x) — Q(x)) + ((H*x)P(*x) +(Vx)Q(x)).

ܘܘܐܐ

*-U2* = -(((*V*x*)P(*x) → (Vx)Q(x)) + (Vx)*(P*(x) →Q(x)))

- the logical equivalence -(*A* + *B) = A1-B* is applied

=*(*(W*x)P(*x)+(Vx*)Q*(x))^-(Vx)(*P*(x) →Q(x)) the logical equivalence *A → B=LAV B* is applied

=*(*-(*Vx)P*(x) v(Vx*)Q*(x))^-*(*Vx)*(P*(x) → Q(x))

• infinitary De Morgan's law: -(Vx) A(x) =(Ex-A(x) and

(*A → B)= A^-B* are applied = ((Ex)*—P*(x) v(Vx*)Q*(x))^(2x)(P(x)^-Q(x)) - rename the bound variables

=((3x)—P(x) v(Wy)2(y))^(Bz)(*P*(z)*^*-Q(z)) the quantifiers (3x), (Vy),(=z) are independent: none of them is within the scope of the other one, so we can extract them in any order. To obtain the simplest Skolem form (with fewest Skolem functions) we extract first the existential quantifiers.

= ()(Ex)(Vy)((-*P*(x)*vQ*(y)) ^ *P(*z)^-Q(z))=*(*-*U2*) - prenex form *(-U*)" = (+x)(Wy)((~*P*(x) V*Q*(x))*^P*(y) ^Q(a)) - Skolem normal form

[z + *a*,x*+b*]*, a,b* - Skolem constants *(U*2) =*(-P(b) v*2(*y*)) ^ P*(a*)^-*Q*(a) - clausal normal form The set of clauses used in predicate resolution is:

S2 = {C'=*~P(b)v*Qly), Cz'*= P(a*), Cz'= -Q(*a)}*

141

Resolution Proof Method

The only resolvent that can be derived is:

Ca'= Resfraj(C',C3')=*~P(6)* The literals *P(a), P(6*) are not unifiable because *a, b* are distinct constants, so the clauses Cz'*= P(a*), C4'=*-P(b*) do not resolve and thus a cannot be derived.

**ses**

**S Ocal**

We proved: (-*U*2) Hres o, so H(Wx)*(P*(x*) →* Q(x)) + ((Vx*)P*(x) → (Vx*)*Q(x))

*U*, is a theorem, but *U*, is not a theorem, therefore we conclude that the quantifier *H is not distributive over the connective →', it is only semi-distributive.* Example 5.19. Using predicate resolution check the validity of the formula:

*U =* (y)(Ex)-*(P*(x, y) H *P* (x,x))

We apply Theorem 5.12 and we prove by contradiction that *U* is valid.

*-U=*-((Wy)(3x)-(P(x, y) ++-*P*(x,x)))=(Ey)(Wx)(P(x, y) +*-P*(x,x)) =

= (By)(Vx)((-P(x,y) v—P(x,x)*)^*(P(x,x*) v* P(x, y))) - prenex normal form *GU*)S = (Vx)(\_P(x*,a*) *V-P*(x,x*))^(P(*x,x) *v P(*x*,a*)) - Skolem normal form,

[x+ a), a- Skolem constant

*GU*) = *(*-*P*(x*,a) V-P*(x,x)) ^ *(P*(x,x) *v P*(x*,a*))- clausal normal form

From the above clausal normal form we consider the clauses:

C = *P*(x*,a) v P*(x,x) and *C2 = P*(x,x) *v P*(x*,a)*

Resolving these two clauses we can derive only tautologies. It is important to apply the factoring rule for C, and C, and try to resolve their factors.

Cite Cz =*-P(a,a)*, Cz is a factor of C. Cz Hotel and C*4 = P(a, a)*, Ca is a factor of C2.

C3,C4 Freso *GU*) Freso, therefore -U is inconsistent and *U is a valid formula.*

142

A Computational Approach to Classical Logics and Circuits

*5.*6. **S*emantic resolution***

Semantic resolution, proposed by J.R.Slagle in 1965 [57], is a refinement of general resolution. In order to reduce the number of irrelevant and redundant clauses in the resolution process, this refinement of resolution:

• divides the set of clauses into two subsets using an interpretation (semantic

aspect), to avoid resolving clauses within the same subset; uses the ordering of propositional/predicate symbols to impose a restriction on the literals resolved upon: the literal resolved upon from the parent clauses contains the largest symbol.

We suppose that the initial set of clauses, S, was simplified: tautologies, subsumed clauses and the clauses which contain pure literals were deleted. S is divided into the subsets: Se and Sy using an interpretation *I* such that all the clauses from Se are falsified by *I* and all the clauses from Sy are satisfied by *I :*

S = *SE U*SN.

There always exists at least one interpretation such that Se and Sy are non-empty sets.

The basic idea is to avoid resolving two clauses which are both falsified or both satisfied by the same interpretation, because the resolvent is irrelevant in the process of deriving the empty clause.

The ordering of the propositional/predicate symbols and the restriction on the literal resolved upon cut down the number of useless clauses generated in the resolution process.

Another important idea is to unify all the variants of generating the same resolvent, where the order of using the parent clauses is immaterial.

**Example 5.20.** Let S ={*Ei = p, E2 =q, Ez =r,N=-pv*-v-r} be a set of propositional clauses. There are six possible derivations of the empty clause, with the only difference being the order of using the clauses *E1, E2, E*z as parent clauses. We present in the following two such derivations of o. *Version 1:*

*Version* 2: *R2* = Res(N*, Ei)=-q*var;

*R'*z = Res*(N, E2)=p*v-*r*; Rz = Res(*R2,E2*)=-8;

R'z = Res(*R2, E3)=-p;* R4 = Res(*R3, E3* ) = 0;

*R*'4 = Res(*R3, E*1) = 0;

143

Resolution Proof Method

To avoid all these versions being considered distinct derivations of the empty clause we shall unify them. The combination of all the ideas presented informally above is formalized in the following definition.

**Definition 5.6.** Let *I* be an interpretation and P an ordering of the propositional*/* predicate symbols. A finite set of clauses: *{E1, E2,..., E,,N},9*21, is called a *semantic clash* with respect to *P* and *I* (or *PI-clash*) if and only if *E*1*, E2 ....,E,* (called *electron*s) and *N (*called *nucleus)* satisfy the following conditions: 1. *E1, E2 ,..., E,* are falsified by *I.* 3. Let *Ri = N* and Ri+1 = Res*p(R;, E), i* = 1,...*,q*, where the literal resolved upon

in *E,* contains the largest symbol in that clause. The resolvents *R, R2,...,Ro,*

are all satisfied by *I. 2. Rg*+1 is falsified by *I. Rg+*1 = Res*pi(N, E1, E2,...,E*,) is called a *PI-resolvent*

of the *PI*-clash *{E1, E2,...,E,*,N*} .*

**Remarks:**

• The order of the electrons is immaterial.

• In semantic resolution we can use any ordering of propositional*/*predicate

symbols and any interpretation. The resolvents *R1, R2,..., Rg*, are all satisfied by *I.* We do not keep them because they will not be used further in the resolution process. All the *Pl-*resolvents are falsified by the interpretation *I,* so they are added to the set of electrons.

**Example 5**.21.

*S = {p,q,r,-p V- V*-r} is a set of propositional clauses, *I* is an interpretation, *1:{p,q,r*} → *{T,F}, I(p)=F, I(9)=F, I(r)=F*, and *P: p>q>* is an ordering of the propositional variables.

The set S is a *Pl*-clash:

• Se *= {E= p, E2 =9, E3 =r*}, the electrons are falsified by *I*

• Sy = {*N=-pv-*v-r}, the nucleus is satisfied by *I*

*Ry =N=pVv* r is satisfied by *I R2* = Res*p(R1,E1)*=-v- is satisfied by *I*

**144**

A Computational Approach to Classical Logics and Circuits

• Rz = Res*p(R2, E2*)=-r is satisfied by *I*

• *R4* = Resp(*R3, E*3)=o=Res*pi(N, E1, E2, E*3) is falsified by *I.* We represent graphically the *Pl-*clash as follows:

N=-*pv*-qv=r E2 = 9

Res*pi (N, E1, E2, E*3)=0

**Exampl**e 5.22.

S = *{E} = p vr, E2 =qvr,*N=*pV v r*} is a set of clauses, *I* is an interpretation, *1:{p,q,r*} → *{T,F}, I(p)= F, 19)=F, I(r) = F,* and *P:r>p>q* is an ordering of the propositional symbols. S is divided by the interpretation *I* as follows:

• S*e ={E} = p vr, E2 =q v*r}, the electrons are falsified by *I*

• Sn =*{N =-pv-v*r}, the nucleus is satisfied by *I* The underlined symbols are the largest in the electrons, according to the ordering *P*. No electrons can resolve with the nucleus, so *S is not a Pl-clash.*

**Definition 5**.7. Let S be a set of clauses, *I* an interpretation of S and *P* an ordering of the propositional*/*predicate symbols of S. A deduction from S is called a *PI-deduction* if and only if each clause in the deduction is either a clause of S or a *Pl-*resolvent.

**Theorem 5.14. Soundness and completen**ess [57] Let S be set of clauses, *I* an interpretation of S and *P* an ordering of the propositional*/*predicate symbols of S. S is inconsistent if and only if there exists a *PI-*deduction of the empty clause from S.

**Remark:**

The completeness property is preserved if this refinement of resolution is combined with the deletion strategy: because no tautologies are generated we

only delete the subsumed clauses. Example 5.23. Using the semantic resolution prove that the following set of clauses is inconsistent. S = *{p,qv-prvp-pv*V r}

145

Resolution Proof Method

1. We choose the interpretation 1 *: {p,q,r} →{T,F}, 1(p)=F, 1(9) = F,*

1*1(r)= F*, and the ordering *P :p>q>r* S*e ={E} = p*}, the electron is falsified by *I* Sn = {*N, =qv-p,N2 =rV-, N*3 = *-PV*-*V*-r}, the nuclei are satisfied

by 11 The *Pi-*deduction of the empty clause is represented graphically as follows:

E;=

Ni=qv-p

E. = p

N2 =rv-p]

Ez = Res *p/(N*,*,E*)=

9

N3 =*-pv*-*qv*ar

Ez = Respi (*N2,E*)=r

*E4* = Res*pi(*N*3, E1, E2,E*3)=0

Th*e Pl-*resolvents *E2, E*3 are falsified by the interpretation I, and they are used as electrons further in the semantic resolution process. 3. We choose: *12:{p,q,*r} →*{T,F},12(p)=7,12(q) = ,12(r)=F*

and *P2:q>r>p* S*e* =*{E1 =qvp, E2 =rvp*}, the electrons are falsified by *12 Sy* = {N, *= p, N2 =-pVqV*mr}, the nuclei are satisfied by *12* The *PI*-deduction of the empty clause is represented as follows:

E =q V-P

N2 =-pv=qv»

Ez=rv-p

*E*z = Res *pi(N2,E1, E2)=p*

E4 = Res p*y (N1,E3*)=0

Special cases of semantic resolution are the s*et-of-support strate*gy [65] and *hyperresolution (*positive and negative) [51].

*The set-of support strategy*: avoids to resolve two clauses belonging to a consistent subset of the initial set of clauses, because the resolvents derived from a consistent set are irrelevant in the process of deriving the empty clause.

**146**

A Computational Approach to Classical Logics and Circuits

***Hyperresolution*** is a semantic resolution with a particular interpretation that assigns the same truth value *(F* or *T*) to all the propositional variables.

**Definition 5.8. A *clause is positi****ve* if it contains only non-negated atoms. A *clause is negative* if all its literals are negated atoms. A *clause* is called *mixed* if it is neither positive nor negative.

**Definition 5.9.** A *positive hyperresolution* is a special case of *Pl-*resolution in which the interpretation *I* assigns to all the propositional variables the truth value *F*. It is called positive hyperresolution because *all the electrons and Pl-resolvents are positive clauses.*

Note: The negative and mixed clauses are nuclei. **Definition 5.10.** A *negative hyperresolution* is a special case of *Pl-*resolution in which the interpretation *I* assigns to all the propositional variables the truth value *T*. It is called negative hyperresolution because *all the electrons and Pl-resolvents are negative clauses.*

Note: The positive and mixed clauses are nuclei. Often, the hypotheses of a deduction are represented by some positive and mixed clauses, and the negation of the conclusion is represented as a negative clause. The *positive hyperresolution* corresponds to “th**inking forward**”: from the hypotheses the conclusion is derived and together with the negation of the conclusion the empty clause is obtained. The *negative hyperresolution* corresponds to **"thinking backward**”: from the negation of the conclusion and the hypotheses the empty clause is derived.

**Example** 5.24. Using positive and negative hyperresolution prove the following deduction:

*qV1,9 →r*,

wr Aw.

We consider the set of clauses corresponding to the hypotheses and the negation of the conclusion: *S*=*{qv1,- Vr*,w,71 V W}

*Positive hyperresolution -* thinking forward The interpretation is 1*1:{9,r*, w} = *{T,F},\|(9)=F,11(r)= F,*I (w)= *F* ordering of the symbols is *P1:q> r* >w.

and the

**147**

Resolution Proof Method

*Se* =*{E, =qvr, E2* = w}, the electrons are falsified by *I.* Sy = {*N*, =*-qur, N2 =* -r V-W}, the nuclei are satisfied by I, The *Pi-*deduction of the empty clause is graphically represented as follows:

E =qvr N =-q vrl

Ez = Respy (N,,E;)=r

N2 =-rv-W

E2 = w

*E*4 = Resp: *(N2, E2, E*3)= 0

*Negative hyperresolution*- thinking backward

S=*{qvr, vr*,w, rv-W}. The interpretation is *12:{q,r*, w} -*>{T,F},12(9)=T, 12(r)=1,12*(w)=*T,*

and the ordering of the propositional variables is P2 : W>*q>r Se ={E=*rVW} the electron is falsified by *12*

*Sn = {N, =qvr,N2 =-qvr,N*z = w} the nuclei are satisfied by *12* The *Pl-*deduction of the empty clause is

E = TV-W N3 = w Ez = Res p*(*N3,E,)= Ez = Res *py*(N, *E*2)=q

N;=qvr] N2=-*q*vr

E2 ==

Ez = Res*pi (N3, E*) ==

N; =*qu*r

*E4* = Respi *(N2,E2,E*3) = 0

*Positive hyperresolution c*an be simulated by lock resolution using the following rules for indexing the literals from the clauses [33]:

**• RP**1: In the positive and negative clauses the indices of literals increase while

the propositional symbols decrease in the ordering P of symbols. **RP**2: In the mixed clauses the indices of literals with the negation sign are smaller than the indices of the literals without the negation sign. For the literals with the same sign, the rule for indexing is RP 1.

**148**

A Computational Approach to Classical Logics and Circuits

These rules block the resolution of two mixed clauses and the resolution of a negative clause and a mixed one. Therefore the resolution is permitted only to a positive clause (electron) and a negative or a mixed clause (nucleus). The literal resolved upon from the positive clause (electron) is the one which contains the largest propositional/predicate symbol.

**Example** 5.25.

S =*{p v q, pvq,PV* 7*9, pV* -9} is a set of clauses and

*P:p>q* an ordering of the propositional symbols.

We shall simulate positive hyperresolution by lock resolution.

*1:{p,g} + {T,F}, I(p)=F, I(9)= F*

The literals from the clauses are indexed according to the above rules as follows:

*E =w pvc*) *9* is an electron, the literal resolved upon must be *p.* The nuclei are the clauses *Ni, N2,N*3 and we apply the rules RP, and RP2

*N*i =(3) *-p V(4)9, N2* =6*) P*V(5) 79, *N3 =(7) P*V (8) 7*9*

**lock resolution:**

**positive hyperresolution:** E = pvq Ni=-*p*va

*N*i=13) *P*V *(4*)9

E-, PV (219 B2-289

*E2=(2)9*

N*2*=*16PV* (5) 7*9*

M. =(3) \*p(479 N=9PY(57 *Nz*=(7) \*PV(8) -9

E2 =9 N2 = pv-g [E = T E =g N = pv-g|

*Ez=(6)P*

N4=(8) 7*9*

149

Resolution Proof Method

***Algorithm positive\_hyperresolution propositi****onal\_logic* (33] **input:** S - a set of clauses, *P* - an ordering of the propositional symbols; **output**: message:"S *is inconsistent*" or "S *is c****onsistent”* begin**

*E*o:= all positive clauses from S indexed according to RP, N°:= all non-positive (negative or mixed) clauses from S, indexed according

to RP, and RP2 *i:=*0; *//* we will generate levels of electrons *E*' and levels of nuclei *N', i*>0 **do**

*i:=i*+1 W := {Reslock (C1,C2)*C, E E-!,C*2 € Ni-"} if (DeW ) then write “S is inconsistent”; e**xit end\_if** *E':=E-*l U{all positive clauses from W}

*Ni:=Ni*-l u{all non-positive clauses from W} } until *( E' = £*1-1 and Ni = *w*i-I) write “S is consistent" **end**

*Negative hyperresolution* can be simulated by lock resolution using the following rules for indexing the literals from the clauses [33]:

• RN,: In the positive and negative clauses the indices of literals increase while

the propositional symbols decrease in the ordering *P* of symbols.

• RN2: In the mixed clauses the indices of literals with the negation sign are

greater than the indices of literals without the negation sign. For the literals with the same sign, the rule for indexing is RN.

These rules block the resolution of two mixed clauses and the resolution of a positive clause and a mixed one. Therefore the resolution is permitted only to a negative clause (electron) and a positive or mixed clause (nucleus). The literal resolved upon from the negative clause (electron) must contain the largest propositional/predicate symbol.

150

A Computational Approach to Classical Logics and Circuits

**Exampl**e 5.26. *S* = {*pvq, pvq,pv , pvq* } is a set of propositional clauses and

*P:q>p* is an ordering of the propositional symbols. We shall simulate negative hyperresolution by lock resolution.

*1:{p,q} + {T,F},I(p)=1,1(q) =T* The literals from clauses are indexed according to the above rules as follows:

*E =*2) *-PV*I) is an electron, the literal resolved upon must be *q.* The nuclei are the clauses N*i, N2,*Nz*:*

*Ni* =(4) *PV3*) *9, N2 =(5) PV (6*) 7*9,* N3 =(8) *PV (7)* For comparison, the negative hyperresolution process and the lock resolution process are represented graphically in the following.

**negative hyperresolution:**

**lock resolution:** *E*i = p V N, = *-pvg* E1=(2) PV (1)-9| N1=(4) \*PV(339

E2=1217P

N2=(15)PV6-9

Ez = -p N2 = pv=4 Ez =-2 E2 =-p] N3 = pvg

*E3=1*679

N3=(8*) PV* (739

*5.7. Exercises*

**Exercise 5.1.** Using general resolution prove that the following formulas are theorems. 1. *V*j =*(-B* →-)*((-B* → A) *+ B*); 2*. U2 =(B-→ A*)^*(C* → *A*) *+(BAC* → A); 3. *Uz =(B → A*)^*(*C → A) → *(BVC* → A); 4. *U4* =(

A C)((A → *B*)+*(-B* TMC)); 5. *Uz = AV(B+C*)+(*AvB)*(AVC); 6. *U6* =(*A + B*)((C → A) → *(*C *+ B*)); *7. U, =(A + B*) + ((A C) → *GB* → C)); 8. *Ug =(A + B^*C) +(A *→B*)^(A →C).

151

Resolution Proof Method

Exercise 5.2. Using lock resolution check the inconsistency of the following sets of clauses. Choose two different indexings for the literals: 1. *S*y ={*pvq, pvzvr,pv*-v-*r,-pvr, PV Gr*}; 2. S2 ={-*PV-9, -pvqvarpv-r,vrpvr*}; 3. Sz =*{pvq,pvqv-r, PVær,r, pvr*};

S4 = *{pvq, pvavar, pvqur, qv-r, vr*}; 5. *S5* =*{pv79,-pv-qur, pvqvrpvq,* 77}; 6. S*6 ={pv9, -pvqv*o*r, PVV-r,pv9,7}; 7.* S7=*{pV9, PV-vr, PVV-r,rvq, rvq*}; 8. S8 = *{pvr,pvqvar, pv-vr, -pvgvr*, ar}. Exercise 5.3. Build a linear refutation from the following set of clauses: 1. Si = *{pvgvr, qvr, -, pvr*}; 2. S2 = {*pvar,qvr, vr, PV*-r}; 3. Sz *= {qvr, -p,*7*9vr,pv*-*r*};

S4={-*pvq,pvvr, r, pvqvi, pv9};*

*S5* = *{pvr, 79, pvqvar, Vr, vr*}; 6. S*o* =*{pvq, pvq-PV , P*V-9}; *7.* Sy *= {p,qvr,-pvqvar, PV-9*}; 8. Sg = {*pvvr,q,-PVvr,-pVqvar,pv*-r}. Exercise 5.4. Prove the consistency of the following sets of clauses. 1. Si = {*pvqr,-qvr, or,-pvr*}; 2. S*2 = {pv-r,qvr, qvr, -pv*-r}; 3. Sz *= {q vr, -p, vr,pv*ær};

*S*4 = {*pvq, PV* 7*9 V1, nr,pvqvr, PV-9};* 5. S5 =*{pvr, mq, pvqv-r,-pv*-*r,qvr};* 6. S*o = {pvq, -pv9,-PV, PV-9*}; *7.* S*7* = *{p,qvr, upvqv*mr, *PV 9};* 8. *Sg* = *{p v vr,q, pv*z*avr, pv*z*vor,pv*r}.

=

0

***1*52**

A Computational Approach to Classical Logics and Circuits

Exercise 5.5. Using the set-of-support strategy prove the following deductions: 1. *(pvq) ►r, pvqvr, 7*+*91-r; 2. pv-r,-q+1,-9*5-*(p+q*); 3. *qar →p, pv9,9+*r+*p;* 4. *r → pvq, p r*,7*9 P1*7*9;* 5. *P→9,*(*9+r)*^-r*tp^-r*; 6. *9 →p,qvr, p +r*+r; *7. P→qur, -9,*

*p a*t*-(pvq)^r;* 8. *r → p-p,9 → pvr*tG*p + qvr).*

Exercise 5.6. Prove the inconsistency of the following set of clauses using lock resolution. Try two different indexings for the literals. 1. S={- *P(*x) *v g(*x)*, p(a), 79*(x)*V*-r(x), -w(*a*), r(y) v w(y)}; 2. S2 = { *p*(x*) V-*(x*), -p(a) vr*(x), *9*(x), w(z), ~r(y) VW(y)}; 3. Sz = { *p*(x*) v9*(x) *v*r(x), *p(a), -9*(x), -w(a), -r(y) v w(y)}; 4. S4 = *{ p*(*x) v9*(x), *p*(x) v r(x), (y) *vr*(y), -r(x) v w(x), -W*( f*(z))} 5. Sz = { *p*(x*)vq*(x)*, -p(a) v* w(x), *9*(y) *vr*(y), -r(x) v w(x), -w(a)};

Sq={*p*(x) *V-9*(x), *p(*z) VW(x),(y) v wy) V-r(y), (x) V-W(x), *r*(g(*a,b)}*

*S7* = {*p(*x) *v g*(x), *p(*x), 7*9(f(a)) v r*(2), -W(2), -r(y) v w(y)} 8. Sg = {-*p*(x*) v g(x) V*-r(x)*, p(f(b)), -9(*x), -W(y), r(y) v w(y)}.

Exercise 5.7*.* Prove the following deductions using linear resolution 1. (Vx)(Vy)(*p*(y,x*)* 4*9*(x*) →q*(y)),(Wx)(Wy)(r(y,x) *→9*(y*)), r(b,a),*

*r(b, a), psc,b*)F(Ez)(z); 2. (Vx)*(p*(x) >r(x)),(Wy)*(*r(y) — 9(y)*), p(a), p(b*) f(32*)q*(z); 3. (Vx)(-*p*(x)^-*9*(x) +r(x)), (Wy)(r(y) → w(y)), (Wx)(w(x) → p(x)),

*p(a), -p(b*), -W(c)F(3z*)q*(z*);* 4. (Vx)(*p*(x) >r(x))*,*(Wy)(r(y) *→q*(*y)),r(a),r(b),* -r(c)(3z)*q(z)*; 5. (Vx)*(~*P(x)^-*9*(x) +r(x))*,*(Wy)(r(y) → w(y)), (Vx)(w(x) → p(x)),

*p*(a), -(c)(*)q*(z);

153

Resolution Proof Method

6. (Vx)(Vy)(-+(y,x) →9(y))*,*(Vx)(Vy)(r(y,x)*^2*(x) →q(y)),

*r(b, a), -p(a,b*) f(*)q(*z); *7.* (Vx)(*P*(x) +r(x)), (Vy)(p(y) +9(*y)), p(a), -r*(c)F(Ez*)q*(z); 8. (Vx)*(p*(x) → r(x)),(Wy)(*p*(y) → 9(y))*, p(a), p(b), -p*(c)(3x)*9(2*).

Exercise 5.8. Using a refinement of predicate resolution prove:

1. the semidistributivity of 'V'over'v':

F(Hx*)p*(x*) v (V*x*)9*(x) → (Vx)*(p*(x*) v 9*(x)) and

H(Vx)(*p*(x*) v g(*x)) → (Vx*)p(*x) v (Vx*)g*(x) 2. the semidistributivity of 3 'over' 1?:

F(x)*(p*(*x)*^*9(*x))+ (3x)*p*(x*)*^(3x*)*q(x) and

H (Ex*)p(*x) ^ (3x*)?*(x) + (2x)*(p*(x) *19(*x)) 3. F(3x)(p(x) 9(x)) *+ (*(Wx*)p*(x) → (3x*)g*(x)); 4. the distributivity of I' over '*V*?:

F(3x)(p(x*) v9(x)*) + (3x*)p*(x) v (2x)q(x); 5. (3x)*p*(x) v (3x)(*p*(x) *19*(x)) + (3x)*p*(x); 6. the semidistributivity of over ':

(Vx*)(p*(x) →9(x)) + *(*(Vx*)p*(x) → (*V*x*)9*(x)) and

H*((V*x*)p(*x) → (Vx*)q*(x*)*) → (Vx)(p(x)9(x)) 7. F(Vx*)p*(x*)*^((Wx*)p*(x) *v* (Vx*)g*(x)) 4 (Vx*)p*(x); 8. the distributivity of 'V' over '1?:

F(*Vx)p*(x*)*^(V*x)*(x) 4 (*V*x)(*p*(x)*^9(*x)*).* Exercise 5.9. Check if the following formulas are theorems using lock resolution. 1. *U*, = (Vx)(V*y)p*(x, y) + (y)(x*)p*(x,y); 2*. U2* = (y)(3x)*p*(x,y) (Ex)(y)*p*(x, y); 3. *Uz =*(Vx)(Vy)*p*(x, y) + (2x)(Vy)*p*(x, y); 4. *U*4 = (3x)(Vy)*p*(x, y) + (y)(x*)p*(x,y); 5. *U*z = (3y)(Ex*)p*(x,y) → (Vx)(3y*)p*(x,y); 6. *U6 =(V*y)(Vx*)p*(x, y) + (Vx)(3y)p(x,y); *7. Un =* (3y)(Ex)*p*(x, y) + (2x)(Vy)p(x,y); 8. *Ug* = (Vy)(Ex*)p*(x,y) H (Ey)(3x)*p*(x,y).

o

n

154

A Computational Approach to Classical Logics and Circuits

**6. REASONING MODELING AND**

**PROGRAM VERIFICATION**

Propositional logic and predicate logic can be used to model simple types of human common-sense reasoning: decide if a statement *(conjecture, conclusion)* is derivable from a set of statements *(hypothese*s). The hypotheses and the conclusion are expressed in natural language. Since predicate logic allows reasoning about the objects of some universe and the relations among these objects, this formalism is also appropriate to model mathematical reasoning. The proof methods presented in the previous chapters are applied in numerous examples.

6*.1. C****ommon-sense reasonin****g modeling using propositional logic*

**Example 6.1. Party** Hypotheses:

*H*: Mary will go to the party if Lucy will go and George will not go. *H2*: If John will go to the party then Lucy will go too. *H3*: If John is in town he will go to the party. *H4*: George is sick and can't go to the party.

*H3:* Yesterday John has returned in town from Paris. Conclusion:

C: Will Mary go to the party*?* We have to check if the following deduction holds.

*H1,H2,H3, H4, H*5-*C*

The following notations for the propositional variables are used:

M - Mary will go to the party *L* - Lucy will go to the party G - George will go to the party *J* - John will go to the party

*Jt* – John is in town

The hypotheses and the conclusion are transformed into propositional formulas as follows:

155

Reasoning modeling and program verification *H:L^*-G→*M H2:1→ Hz:Jt →] H4*:46

*H5:Jt*

*C:M*

The definition of the deduction (Definition 1.8) and the axiomatic system are used:

*fi = H,:L*A-

G M (hypothesis) *f2 = H2:J →L* (hypothesis) *f3 = Hz:Jt → J* (hypothesis) *f4 = H4:*-*G* (hypothesis) *fs = Hş: Jt* (hypothesis) *$5,f*3m*p Jifo fo, f*3 Em*p L:57 $4,8*7E*L*A-*G:fo (*conjunction in conclusions)

*f&,* fi tm*p M: f9=C* M*odus ponen*s inference rule was applied to derive the formulas *fo, ft,fg.*

The sequence of formulas: *(81, 82, 83,54,55,56,87,*8*8,fg*) is the deduction of C from the hypotheses, therefore, based on the hypotheses*, Mary will go to the party.*

**Example 6.2.** Hypotheses:

*H:* If it is sunny, Diane and Alice go to the swimming pool. *H*2: Ben goes to the swimming pool on Thursdays.

*H*z: It was sunny last Thursday. Conclusion:

*C* : Did Diane meet Ben at the swimming pool last Thursday? We have to check if the following deduction holds:

*H1,H2,H3-C* The resolution, a refutation proof method, is applied and according to the Theorem of soundness and completeness of resolution, we have to check if:

CNF*(H, ^H2 ^ H3*1-C)FRes 0.

We use the following notations for the propositional variables:

*D -* Diane goes to the swimming pool *A* - Alice goes to the swimming pool *B* - Ben goes to the swimming pool S - it is sunny *Th* - it is Thursday

156

A Computational Approach to Classical Logics and Circuits

The following propositional formulas and their conjunctive normal forms correspond to the hypotheses and the negation of the conclusion:

*H*:*S →DA*A=*SV(DAA*)=(*SVD)^*(-S v A)*:C1C*2 *H2:Th → B=-Th v B:Cz Hz:S ^ Th:C44C5 C:ThADAB* -C:*-(Th^D^B)=-ThVDVB:Co*

The hypotheses and the negation of the conclusion were transformed into CNFs and the set of clauses S = {*C*1,C2*,C*3,C4,*C*5,C6} was obtained:.

If we want to apply the set-of-support strategy, we have to avoid resolving two clauses belonging to the consistent subset of clauses: {*C*1,*C*2,*C*3,C4,C5} provided by the hypotheses. The support set of s is Y = {C} and corresponds to the negation of the conclusion. All the resolvents are added to Y. The refutation from S (the derivation of o from S) is provided below:

C1 = Res(C5*,*C6*)=-DV-B* Cg = Res(C7*,C*3)=-*DV-Th* Cg = Res(C3,C3)=-*D* C10 = Res(Cg,G)=-S Cu = Res(C10,C4)=0

We have proved that CNF*(H 1H,1H3*1-C)FRes a, so C is deducible from the hypotheses, therefore: “*Diane met Ben at the swimming pool last Thursday".*

We shall prove the deduction *H,H2,H*3tRes C using another proof method, a direct method, the sequent calculus. The up-side down binary tree corresponding to the reduction process of the initial sequent *H1,H2,H3C* is built.

The initial sequent was reduced to five basic sequents (overlined), therefore it is a true sequent and according to the theorem of soundness and completeness of this method we conclude that C is deducible from the hypotheses *H,H2,Hz.*

157

158

*D, A, B, S,Th =D B, S,ThS,D DIA, B, S,ThD*

- >)

S → *DNA, B, S,Th=B S →DA A, B,S,Th =D S → DNA, S,Th Th, DAB* S -> *DNA, B, S,Th DAB*

-*(1*) *for Th → B ..*..

S *→DA A, Th → B, S,Th=DAB*

S *+ DA A,Th • B, S,Th 4 Th*

S *→ DA A, Th → B,* S*ATh=DAB Ath*

**Example 6.3.** A client describes the requirements of a software application:

*Ry*. If condition A is satisfied then condition *B* must also be satisfied. *R2*. If conditions B and C are satisfied, then *D* must also be satisfied. *Rz*. If condition *D* is satisfied then condition A is not satisfied, *Rd*. If condition C is satisfied then A must also be satisfied. *Rs*. If A is satisfied then *D o*r C are satisfied. *Rs. C* is satisfied if neither B nor A are satisfied. *Ry. B* is not satisfied if C is not satisfied.

Reasoning modeling and program verification

Are these requirements simultaneously satisfiable? In order to answer the question we have to check the consistenc*y/*inconsistency of *U = Ri 1 R2 ^ R3* ^ *R4 ARS ARE* A *Rz* The statements corresponding to the requirements are transformed into propositional formulas and further into their conjunctive normal forms:

A Computational Approach to Classical Logics and Circuits

*R1. A+B* = *-AVB: G*. *R2. BAC →D = BV-CvD: C2 Rz. DA*=-*D*VGA: *C*3 *R4.C* → A = -C *V*A: *C*4 *Rs. A CvD* = -*AVCvD: Cg Rr. A1-*

*B C* = *Av BvC: Co Ry.* -C → *B = CV-B: Cy* S = {C1,C2,C3,*C*4,*C5,Co,C7*}. We have to check the consistency/inconsistency of the sets of clauses corresponding to the conjunction of the requirements. The general resolution procedure is applied:

Cg = Res(C1,C7)=*C*VA Cg = Res(C7*,C*6)=*CVA* C10 = Res(Cg*, C*y)=*C* Cu = Res(C4,C1o) = A *C*12 = Res(C1*,C*)=*B* C13 = Res(C11*,C*3)=-*D* C14 = Res(C13*,*C2)=-*BV*-C C15 = Res(C14,C12)=-C *C*16 = Res(C10,C15)= 0

The empty clause was derived from S, so S *is inconsistent and the requirements are contradictory.* Resolution as a proof method is very efficient compared to the truth table method as we shall see in the following. To check the consistency/inconsistency of the requirements *Ry – R,* we have to build the truth table of the propositional formula representing the conjunction of all the requirements: *U = R1 R2 ^ Rz* 4 R*4* ^ R*s* ^ *Ro^ Ry.*

Each requirement has a conjunctive normal form composed of one clause (see the normalization process presented before), thus:

*U* =Ci*nC2*-C3-C44*C*54*C*61*C7*.

The formula U has 4 propositional variables: *A,B,C,D,* so its truth table has 24 = 16 rows, corresponding to all 16 interpretations which assign in all the possible ways the truth values *{T,F*} to the variables *A,B,C,D.*

159

Reasoning modeling and program verification

*т*

***F***

*T*

*т*

TL *T*

*T*

|

A | *B* | C | *D* | *R* | R2 *R3 RA Rs* | Ro | *R U* i*TTTTTTTTTT*T*T F*.

*TTT FT F*I*TT Τ* Τ Τ Τ Τ Τ *F*

*T* 1

*T* 1 *F*I*T T* 1 Τ Ι *F*I TI *T T*I*F т F F*

*T*

Τ Τ Τ Τ Τ Τ *F* TI *F т*

*T* | *F*I*T*

*т T FTF F*T

*Τ*

*T*T*T*T *TFF TF*

*TI T*I*F* i til*f*t*e*telliti t*i*ttit Lig *FIIIIII T F* L *T Τ* Ι ΤΙ *F*

to F Ι Τ Ι Τ Ι FI TI *F*

*Τ* Τ Τ Τ Τ Τ *F F I F T I*

***T***

*TIFF 1*1*2 | F* | *T* | *FIFT T* TITI

*Τ* Ι Τ Ι FI *F* 13T *FI F*I Τ Ι Τ *Ι* Τ Ι Τ Ι Τ Ι F Ι Τ Ι Τ Ι ΤΙ *F*

i 1 *F F T F. ТТ*

*т т т Е T*I*TT F* 115 *F FFT ITT* I*TT* I *F T I F* 116 *F* | *F* | *F* | *F* | Τ Ι Τ Τ Τ Τ Τ Τ Τ Τ *F* I T I *F* Each requirement is evaluated in all 16 interpretations and the column (truth table) *o*f *U* is obtained as the conjunction of the columns of the requirements. For each row (interpretation) there is at least one value *F* in the columns of *R; - R*y, meaning that in each interpretation at least one requirement is evaluated as false, so *U* is evaluated as false in that interpretation. The column of *U* contains only the truth value *F, s*o *U* is evaluated as false in all 16 interpretations, therefore *U* is an inconsistent formula. The conclusion is that *the requirements are contradictory, they cannot be satisfied simultaneously by the software application.* Note that the truth table method is time consuming and it is not efficient at the implementation level.

**Example 6.4.** Consider the following hypotheses:

*H.* Mary will go to London this summer if both her friends Kate and Susan go. *H*. If Kate passes the English exam in May then she will go to London. *H*z. Kate was in hospital from April until July and she didn't take the English

**exam.** *H4*. This summer Susan will go to London on a business trip.

and the conclusion: *C*. Will Mary go to London this summer?

160

A Computational Approach to Classical Logics and Circuits

is a logical

Using the semantic tableaux method check if the conclusion C consequence of the set of hypotheses: *{H1,H2,H3, H4}.*

We transform the hypotheses and the conclusion into propositional formulas:

*H:K*AS →M *H2:KE →K Hz: -KE H4:S*

*C:*M where the notations for the propositional variables are as follows:

M - Mary will go to London S - Susan will go to London *K* – Kate will go to London

*KE* - Kate passed the English exam The semantic tableau method is a refutation proof method, thus we have to negate the conclusion and use the Theorem of soundness and completeness:

*H,H2,H3, H4*+*C if and only if Hi ^ H2 ^ H3 ^ H4*1-C has a closed semantic tableau.

The semantic tableau corresponding to the conjunction of the hypotheses and the negation of the conclusion is depicted below.

*(K*AS → M)*^(KE → K*AK*E* A*S*AM (1)

J-a rule for (1) *K*AS →*M (*2) *KEK* (3)

*LKE*

M

*--ß* rule for (2) (4)*-(K*ÁS) M *B* rule for (4) —–

closed branch

*K* S *B*rule for (3) —

closed branch *KE* open brancho

closed branch

**161**

Reasoning modeling and program verification

UL

We have obtained a complete and open tableau with one open branch (the left-most one) and three closed branches containing the following pairs of opposite literals: (*K,*K),(S, S),(M,-M)*.* Therefore *H1,H2,H3, H4*#Cand based on the hypotheses we conclude that *Mary will not go to London this summer'.*

6*.2.* ***Reasoning modeling using predi****cate logic*

**Example 6.5.** Hypotheses:

*H :* All hummingbirds are richly colored. *H2:* No large birds live on honey. *Hz*: Birds that do not live on honey are dull in color.

*H4: Piky* is a hummingbird. Conclusions:

*D*i: All hummingbirds are small.

*D2 : Piky* is a small bird and lives on honey.

We have to check if the following deductions hold or not.

*H1,H2,H3* F*D*, an*d H1,H2,H3, H4*F*D2*.

The natural language sentences are transformed into predicate formulas. We use the following unary predicate symbols to express properties of the objects from the universe of birds.

*hb(*x) - x is a hummingbird, *rc*(x) – x is richly colored *sb(*x) – x is a small bird *Ih*(x) – x lives on honey *Piky* is a constant of the universe.

*h*(x*)*)

*H:*(Vx*)(hb(*x) > rc(x)) *H2*:-(3x)( *b(x) Alh(*x*))* =(Vx)(*sb(*x) *H3* :(Vx)*(-1)(*x) → -rc(x)) *H4: hb(Piky)*

*Di* :(Vx)*(hb(x)* → *sb(*x*)*) *D2 : sb(Piky) ^ Ih(Piky)*

162

We prove the deduction *H1,H2,H3* H*D*, using the sequent calculus method, a direct and syntactic proof method. The reduction process of the initial sequent *H,H2,H3 3D*, is depicted below:

*re(t), H1,H2,H3,hb(t), lh(t)= lh(t), sb(t) ro(t), H1,H2,H3, hb(t)=rc(t), lh(t), sb(t) rc(t), H1,H2,H3,hb(t) = -lh(t), lh(t), sh(t) rcst)*, -*rc(t), H,H2,H3, hb(t)= Ih(t), sh(t)*

"? *(+ for C*)

*rc(t),C,H1,H2,H3, hb(t)=lh(t), sh(t) rc(t),C,H,H2,H3, hb(t), sh(t) sb(t) r(t), -1h(t),C,H1,H2,H3, hb(1) sb(t)*

- Gr)

abe? *re(t),C,H1,H2,H3, hb(t)=-sh(t), sb(t)*

*(→ for B)*

*B,C,H1,H2,H3, hb(t)= hb(t), sb(t) r(t),B,C,H1,H2,H3,hb(t)=sb(t*)

*"? (7 for A)*

*A,B,C,H,,H2,H3, hb(t) = sh(t) hb(t) → rc*(*t), -sb(t) → Ih(t), -Ih(t)* → *rc(t), H1,H2,H3 3 hb(t) + sh(t)*

(V)(x+ t,y*tt,z ft*]

*(Vx)(hb(*x) + rc(x)),(Vy)(-*sb(y) → -1h(y*)), (*V*2)*(–1h(z)* → -*r*c(z)) *= hb(t) → sh(t)*

- (V) (*Vx)(hb(*x) → *rc*(x)),(Wy)(-sb(y) →*-lh(*y)), (*Hz*)*(-1h(z)* → -*r*c(z)) = *( t)(hb(t) → sh(t)*)

A Computational Approach to Classical Logics and Circuits

-

(

,)

•

the bound variables from the formulas *H,H2,H3, D*, were renamed such that they are distinct in the initial sequent:

*H :(Wx)(hb(x) → rc*(x)), *H2 :*(Vy)(-*sb(*y) +*-Ih(y)*), *Hz :*(*H*z)*(-1h(z*) +-rc(z)), *Di :(*V*t)(hb(t) → sh(t*)*)*

163

Reasoning modeling and program verification

in the first step the universal quantified variable t from the consequent becomes a free variable: the rule (V.) is applied; in the second step the rule (V) is applied three times. All three universal quantified formulas *H,H2,H3*, from the antecedent, are instantiated using *t* and A*,B,C* open formulas are obtained. *H,H2,H3* are duplicated (for further instantiations);

*A= hb(1) ►rc(t), B = -s(t) -ht),*

*C =-lh(t) → -rc().*

• in the third step the rule (,) is applied;

• in the following steps the formulas A*,B,C* are decomposed using the rule

(+);

• the complete reduction tree has four leaf nodes containing basic sequents.

The initial sequent *H1,H2,H3 = D*was reduced to four basic sequents, so it is a true sequent and the deduction *H1,H2,H3 + D*, holds. Based on the hypotheses we conclude that:

*‘All hummingbirds are small'.*

The deduction *H1,H2,H3, H4* + *D*2 is proved by contradiction by applying the semantic tableaux method. The semantic tableau of *Hi ^ H21 H3 ^ H41-D*2 is built. The copies of the universal quantified formulas *H,H2,Hz a*re not used further because no new constants were introduced after their instantiations.

The semantic tableau is closed, having five closed branches. The set of the hypotheses is consistent, therefore there is a contradiction among the hypotheses and the negation of the conclusion. According to the reduction ad absurdum principle, the conclusion D2 is

derivable from the hypotheses *H,H2,H3, H4.*

Based on the hypotheses we conclude that:

*Piky is a small bird and lives on honey.'*

164

A Computational Approach to Classical Logics and Circuits

*H,1H2 Hz 1H4^\_D2* (1)

|-generalized a rule for (1) *H :(Vx)(hb(*«) ►rc(x))(2)

*H2*:(Vx)(*-sb(*x) +*-lh(*x)) (3)

*H3 :(*Vx)(*-1h(*x) ---rc(x))(4)

*Ha: hb(Piky) -D2 :-(sb(Piky)' \ Ih( Piky*))(5)

-y rule for (2)*, Pik*y used for instantiation *hb(Piky) -> rc(Pik*y) (6)

*H,* -copy of (2)

*- B*rule for (6) *-hb(Píky) rc(Piky)* closed branch

| -y rule for (4*), Pik*y used for instantiation *-lh(Piky*) → *rc(Piky*) *(7*)

*Hz* -copy of (4)

*B*rule for (7) *Ih(Piky) rc(Piky)*

closed branch 1-y rule for (3)*, Piky* used for instantiation *-sb(Piky) +-lh(Piky)* (8) *H2:*(Vx)(*sb(*x) = *-lh(*x)) - copy of (3)

*B*rule for (8) *sb(Piky) -lh(Piky) B* rule for (5)

closed branch *-sb(Piky) -lh(Piky)* closed branch

closed branch

**Example 6.6.** Consider the following set of hypotheses *{H1,H2,H3, H4, H5, H6*} and check the validity of the conclusion (C):

*C. Scroog*e is not a child.

165

Reasoning modeling and program verification

*H* . Every child loves *Santa. H2. E*veryone who loves *Santa* loves any reindeer. *Hz. Rudolph* is a reindeer, and *Rudolph* has a red nose. *H4.* Anything which has a red nose is weird or is a clown. *Hz.* No reindeer is a clown. *H6. Scroog*e does not love anything which is weird.

*H1,H2,H3, H4, H5, H6 FC.* The natural language sentences are transformed into predicate formulas:

*H:*(Vx*)(child(x) + lov*es(x*, Santa)) H2 :*(Wx)(Vy)*(loves(x, Santa) reindeer*(y) *→ loves*(x, y)) *Hz : reindeer(Rudolf) ^ red\_nose(Rudolf) H4:(V2)(red\_nose(*z) → w*eird(z) v clown(*z*)*) *Hz:(*Vs*)(reindeer(*s)→*-clown*(s)) *H6:(Vt*)(w*eird(t)→ loves(Scrooge,1))*

*C:-child(Scrooge)* where:

x,*u*,y,*z,s,t* are bound variables, which will become free variables during the

normalization process, *Rudolf , Santa* and S*croog*e are constants, *child, reindeer, red\_nose, weird, clown* are unary predicates and *lov*es is a binary predicate

We apply a refutation proof method: general predicate resolution. The clausal normal forms corresponding to the hypotheses and the negation of the conclusion are as follows:

*Ho:-child(x) v lo*ves(x*, Santa)=C H2o :-loves(x, Santa) V-reindeer*(y) *v loves*(x, y)=*C2 Hzo : reindeer(Rudolf) A red\_nose(Rudolf )=C*31*C4 H4° :-red\_nose*(z) v w*eird(z) v clown*(z)=C5 *HC :-reindeer*(*s) v-clown*(s)=C6 *H. :*-w*eird(t) v-loves(Scrooge,t)=C,* (-C)*° :child(Scrooge*)=*C*g

166

A Computational Approach to Classical Logics and Circuits

To the set of clauses S =*{C,*C2*,C*3,*C*4*,C5,C6,*C1,Cs}, general predicate resolution is applied. In the resolution process, the following resolvents are obtained:

Cg = Res[xx-Scrooge]*(*C3,C*) = lo*ves*(Scrooge, Santa)* Co = Res*-Rudolf* (CB,C)=*-clown( Rudolf)* Cu = Resez-*Rudol*f ]*(C*4, C3) = w*eird(Rudolf) v clown(Rudolf)* C12 = Res Pr (C10*, C*u)= w*eird(Rudolf)* C13 = Resfit*Rudolf (C*12,Cy) = -*lo*ves(S*crooge, Rudolf)* C14 = Res[y*-Rudolf*]*(*C2,C3)=*-lov*es(x*, Santa) v lo*ves*(x, Rudolf )* Cis = Resort-Scrooge](C13*,*614)=*-loves(Scrooge, Santa)*

C16 = Res" (C9, C15)=0 The most general unifier generated during the resolution process is the substitution:

[x+ S*croog*e, y*r Rudolf ,z + Rudolf ,s + Rudolf ,t+ Rudolf]* Stresa, therefore S is an inconsistent set and the deduction

*H1,H2,H3, H4, H5, H6* EC holds.

From the hypotheses we conclude that "S*crooge is not a child'.*

**Example 6.7*.* Succession to the British throne** Hypotheses:

*H:* If x is the king and y is his oldest son, then y can become the king. *H2*: If x is the king and y defeats x, then y will become the king. *Hz: RichardIII* is the king. *H4: Henry VII* defeated *RichardIII .*

*Hz: Henry VIII is Henry VII's* oldest son. Conclusion:

*C*: Can *HenryVIII* become the king?

Check if the conclusion C is derivable from the set of hypotheses *{H1,H2,H3, H4, H5* } using a syntactic proof method. We transform the hypotheses and the conclusion into predicate formulas using:

variables: x, y,z*,t* constants: *Richard III, Henry VII, Henry VIII* predicate symbols: **unary: *kin*g , binary***: oldest \_son, defeat*

167

Reasoning modeling and program verification

*H:*(Vx)(Vy*)(king(x) ^ oldest\_son*(y,x*) → king*(y)) *H2:*(Hz)(V*t)(king*(*z) ^ defeat(t,z) → king(t)) Hz: king(Richard III) H4:defeat(Henry VII, RichardIII) Hz: oldest\_son( HenryVIII, HenryVII) C: king(HenryVIII)*

We prove the deduction *H,H2,H3, H4, H5 FC* using the axiomatic system of predicate logic and Definition 2.2.

The following sequence of predicate formulas: *(f1, f2,..., $*13) is generated. The inference rules used in the deduction process are **universal instantiation:** *univ\_inst* and **modus ponens*:*** *mp. fi = H, :(V*x)(Vy)(*king(x) ^ oldest\_son*(y,x) *→ king*(y)) *f2 = H2 :*(Vz)(V*t)(king(2) 1 defeat(1, 2*) *→ king()) fz = Hz : king(Richard III) fd = H4: defeat(HenryVII, RichardIII) S5 = Hg: oldest \_son(Henry VIII, Henry VII)*

*$2* Funiv i*nst (t)(king(RichardIII) A defeat(t, Richard III) → king(t*)):*fo,*

the universal variable z was instantiated using the constant *RichardIII* fot univ i*nst king(RichardIII) ^ defeat(HenryVII, RichardIII) king(HenryVII): f7*

the universal variable *t* was instantiated using the constant *HenryVII f3^ $4 = king(RichardIII) A defeat(Henry VII, RichardIII): fs* (conjunction in

conclusions) *f8, f*7 Fm*p king(HenryVII): f, f*ituniv ins (Vy*) (king(HenryVII) oldest \_son(y, HenryVII) → king*(y)): fio,

the universal variable x was instantiated using the constant *Henry VII* fio Funiv*inse king(HenryVII) ^ oldest \_son( HenryVIII, HenryVII) → king(HenryVIII): fu*

the universal variable y was instantiated using the constant *Henry VIII fo^ fg = king(HenryVII) ^ oldest \_son(Henry VIII, HenryVII): f*12 (conjunction in

conclusions) fio, f12 Fm*p king(HenryVIII):* f13=C The sequence of formulas *(*$1*,$2...,* $13) is the deduction of C from the hypotheses *H1,H2,H3, H4, H5,* therefore based on the hypotheses we conclude that ‘*HenryVIII can become the king'*

168

A Computational Approach to Classical Logics and Circuits

**Example 6.8.** Hypotheses:

*H:* Any Computer Science student likes *logic* and likes any programming

language. *H*: Someone who like*s logic* is a Computer Science student or a Philosophy

student. *Hz: Java* is a programming language.

*H4: John d*oesn't like *Java* but he likes *logic.* Conclusion:

*C*: John is a Philosophy student but he is not a Computer Science student. The hypotheses and the conclusion are translated into first-order language. Symbols used:

- x,y,z are variables; *- logic, Java, John a*re constants, - CS and *P* are unary predicate symbols with the meanings:

*CS*(x):' x is a Computer Science student'

*P*(x) :' x is a Philosophy student' *- pl* is a unary predicate*, pl(*x): 'x is a programming language' *- likes* is a binary predicate*, likes*(x, y): 'x likes y'

*H;*:(Vx)(Vy)(CS(x) *^ pl(y*) *→ likes*(*x,logic) ^ likes*(x, y)) *H2 :(*V*z)(likes(z,logic) → CS*(z) *v P(*z*)*) *H3 : pl(Java) H4: likes(John, l*ogic) ^ *likes(John, Java)*

*C*:-CS*(John) A P(John)* We transform the hypotheses and the negation of the conclusion into clausal normal forms: *(H*) = -CS(x) v *pl*(*y) v likes*(*x, logic) 1 likes*(x, y) =

=(-*C*S(x) V *pl*(*y) v likes(x, logic)*)^(-CS(x) *v pl(y) v like*s(x, y)) = C1 AC2

*G*=-CS(x) *V-pl*(*y) v likes(x,logic),* C2=-CS(x) *V-pl*(y*) v likes*(x,y) *(H2*) *= likes(z,logic) v* CS(z) *v P*(Z) = Cz *(Hz)° = pl(Java)*=C4 *(HX) = likes(John, logic)^-likes(John, Java)* =C*3 1 C*o

*Cs = likes(John,logic), Co=-likes(John, Java)* (C) = *CS(John) v—P(John) = C7*

**169**

Reasoning modeling and program verification

**ar re**

We apply linear resolution to the set S={C1*,C2,*C3*,C*4,*C*5,*C6,* C7} of clauses, with Cz as the top clause. Cg = Res x-John}(C3,Cs*) = CS(John) v P(John)* Cg = Res(C3,C7) = *CS(John)* Cho = ResExtJohn *(C*5,*C2)=-pl(y) v likes(John,* y) Cu = Respyt-JavajcC1*0,Ca)= likes(John, Java)* C12 = Res(C1*,C6*)=0 The linear refutation process is represented graphically as follows:

1 [z*t John]*

1 [x+*- John]* Co Ca

[yt *Java]*

C12 =0

*S* hes o, therefore s is an inconsistent set and based on the hypotheses we conclude that:

*John is a Philosophy student but he is not a Computer Science student'.* **Example 6**.9. Reasoning modeling in geometry using predicate logic The domain is the set of all the lines in a plane. We use variables: x, y, z to denote arbitrary objects (lines) and constants: *d,d,,d*, to denote constant objects (lines). Hypotheses:

*H:*If x is perpendicular to y then x intersects y. *H2:* If x is parallel to y then x doesn't intersect y. *H3:* If x is perpendicular to y and z is perpendicular to y then

x is parallel to z.

*170*

A Computational Approach to Classical Logics and Circuits

*H4: d,* is perpendicular to *d.*

*Hz: d* is perpendicular to *dy.* Conclusion.

*C:d2* does not intersect *d.*

the set of hypotheses

Check if the conclusion C is derivable from *{H1,H2,H3, H4, H*5} using a syntactic proof method.

In order to translate the hypotheses and the conclusion into first-order formulas we use the binary predicate symbols with the same names as the corresponding geometric relations. Distinct names for the bound variables in the first-order formulas are used.

*H;*:(Vx1)(Vx2)*(perpendicular*(x*1*,x2)*→ intersects*(x1, x2)) *H2:*(Vx3)(Vx4) *(parallel(x*3, x4) + *intersects*(x3, x4) *H3 :*(Vxs) (Vx6)(Vxz) *(perpendicular(*X5, X6) *^ perpendicular* (X7, X6) *► parallel(x5*, X7)) *H4: perpendicular(d,,d) Hs: perpendicular(d,d2) C: intersects(d2,d*)

We have to add predicate formulas which express the properties of s*ymmetry* for the geometric relations: *parallel, perpendicular, intersects.*

*P*1 :(Wxg)(Wxg)*(parallel(x*8, xg) → *parallel(x9*, xg)) *P2* :(Vx1o) (Vxjj)*(perpendicular(*x10, X11) *► perpendicular(*x11,410)) *P*z :(Vx12)(Vx13*)(intersect*s(x12,813*) intersect*s(x13, X12))

The properties of *reflexivit*y for *parallel* an*d intersect*s, an*d transitivity* for *parallel* must be also added: *P*4 :(Vx14*) parallel(x*14,414) *P*s :(Vx*i5)intersect*s(x15, X15) *P*G:(VX16)(Vx17)(Vx17)*(parallel (*x16,8*17) ^ parallel*(X*17*, X18) *► parallel(x*16,818))

The clausal normal forms of the hypotheses: *H1,H2,H3, H4, H5*, the negation of the conclusion C and the properties: P*1, P2, P3, P4, P5, P*o are as follows: *Ho: -perpendicular(*x1, x*2) v intersects(*x1, x2):C *H2o :-parallel(x*3, x4) *V intersects*(x3, x4):C*2 Ho :-perpendicular*(X*5*, X6) *V-perpendicular(*x7, X6) *v parallel(X5,X7)*:Cz *H4° : perpendicular(dı,d):C*4

171

Reasoning modeling and program verification

*Ho : perpendicular(d,d2):C5* (-C*)o: intersects(d2,dy):Co* po *:-parallel(*x*8,* xy*) v parallel(x9,* xg*)*:C, *po: -perpendicular*(x10, Xu*) v perpendicular*(x11,X10):C8

*pzo: -intersects*(x12,813*) v intersect*s(x13, X12):C, *P&C : parallel(*x14, 814):C10 *PC : intersects*(x15, X15):C11 P*oC : -parallel(x16*, X1*7) v-parallel(x17*,818) *v parallel(*x*16,* X18):C12

Pr

We apply general resolution to the set of clauses: S={C1*,C*2,...,C12}. The following resolvents are obtained during the resolution process: C13 = Resmior-d, xi,*td>*]*(*C5*,C8)= perpendicular(d2,d)* C14 = Respost-d*j*-xotdj*(*C3,C4)*=-perpendicular(x7,d) v parallel(dı,xn*) Cis = Resx+d,j*(*C3*,C*14) *= parallel (dı,d))* Co = Resfestd1,884-dz](C5,C2)*=-intersects (dı, d)*) Cız = Respiratdı.xi*ptd, (*C16*,C*9) *= intersects (dz,di)* C18 = Res(C17,C6)=0

The empty clause was derived from the set S of clauses, so S is inconsistent and the deduction *H1,H2,H3, H4, H5*EC holds. The conclusion '*d, does not intersect di’* is valid, based on the validity of the hypotheses.

**Example 6.10**. Mathematical reasoning modeling in algebra. Prove: “**If every element of a group G is its own inverse, then G is an Abelian group”.**

We will introduce the axioms*, H,H,* which define the group, the hypothesis, *Hz:* "every element of the group is its own inverse”, and the conclusion C : “the group is an Abelian group”.

*Mathematical language:*

*H:*(Vx)(Vy) (Hz)[(x\* y) \* z = x\*(y\* z)] - associativity *H :*(Vx)[x\*e=e\* x = x] - e - neutral element

172

A Computational Approach to Classical Logics and Circuits

*Hz:*(Vx)[x\* x = e]- every element is its own inverse *C:*(Vx)(Vy)[x\* y= y\* x] - conclusion: G is an Abelian group

*First-order logic language:* We use the ternary predicate symbol *P*, with the meaning: P(x, y, z):"x\* y=z”. 'e' is a constant. The formulas U, and *U2* correspond to *H; : U*;:(Vx)(Vy) (V2)(*Vu*)(Vv)(Vw)[*P*(x, y*,u) ^ P(u,z*,w) *^ P*(y,z,v) → *P*(x,v,w)] *U2:*(Vx)(Vy) (Hz)(V*u*)(Vv)(Ww)[P(y,z,v) *^ P*(x,v,w) ^ *P*(x,y*,u) → P(u,* z,w)]

*U" :-P*(x,*y,u) V-P(U,*2,w) V-*P*(y, z,v) *v* P(x,v,w)=C

*U2° :—P*(y,2,V) v*—P*(x,v,w) V-P(x*, y,u) v P(u,*z,w)=C2 The formulas *Uz* and *U*4 correspond to *H2: U*z :(Vx*)P(*x,e,x),

*Uzo:P*(s*,*e,s)=Cz *U4 :*(Vx*)P*(e, x,x),

*UA? :P*(e*,r,r*)=*C*4 The formula *Us* corresponds to *Hz: Us :(Vx)P*(x,x,e),

*1,e*) =C5 The formula *Uo* corresponds to the conclusion: *UG*:(Vx)(Wy)(F*t*)(P(x, *y,t) → P*(y,*x,t*))

*UG*:-((Vx)(Vy)(E*t*)*(*P(x, *y,t)→ P*(y,*x,1))*) =

= (3x)Ey) (V*1)(P*(x, y*,t)^—P*(y*,x,t)) (U6*) *: P(a,b,t)^-P(b,a,t) = Co^C, a,b* – Skolem constants In the clauses we ranamed some of the free variables. Checking whether *H1,H2,H3* +C was reduced to checking the inconsistency of the set of clauses: S ={G*,*C2,C3,*C4,*C5*,C*6*,C;*}. The refutation from S is presented below. At each application of the resolution rule, the literals resolved upon from the parent clauses are underlined.

C=*—P*(x, *y,u) v—P(u, z*,w)v—P(y, z,v) v P(x,v,w), *Cs = P(1,1,e)*

=[x+*1,yt 1,ut* e]*=mgu(P*(x, y*,u), P(1,1,e*))

• Co = Res. (C,C3*)=—*P(e,z,w)*v —P(1, 2,*v*) v P(1,*v,w) *C2 = -P*(y, z,v) *v —P*(x,v,w) *v —P*(x, y*,u) v P(u,z*,w), *Co = P(a,b,t)*

02 =[x + a,y*<b,u+ 1*] *= mgu(P(a,b,t), P*(x, y*,u)*))

• Co = Res. (C2,C6)*=-P(b,*z,v) *v —P*(a,v,w*) v P(t,* z,w)

173

Reasoning modeling and program verification *C4 = P(e,r,r), C*g=—P(e,z,w)v*—P(1, z,v) v P(1,*v,w)

*0*3 = [ztr,

w r]*= mgu(P*(e, z, w)*, Ple,r,r*))

• Cho = Res (C4,C3*)=P(1,r,v) v P(1,v,r) Cs = P(1,1,e), Co*=*-P(b,z,v) v—P(a*,v,w*) v P(t,*z,w)

*04 = [1 + b,z+b*,vt e]*=mgu(P(1,1,e), P(b,* z,v))

,Co)= *P(a, e*,w) *v P(t,b*,w) *Cz = P(s,*e*,s),* Cu*=-P(a,* e,w) *v P(t,b,*w)

*05* = [s*ta*,wt *a*)*= mgu(P(*s,e,s)*, P(a,e*,w))

• C12 = Respo (C3,C11*)=P(t,b,a),* Co=*-P(1,r,v) v P(1,v,r), C12 = P(t,b,a)*

*06 =[l+ t,rt-b*,v*ra]= mgu(P(t,b,a), P(1,r*,v))

• C13 = Res (C10*,C,2)= P(t,,b) C2=-P*(y,z,v) v —*P*(x,v,w) V-P(x,y*,u) v P(u,*z,w), *Cş = P(1,1,e)*

*0* =[y +*1,24 1*,vt e]*=mgu(P(1,1,e),P*(y,z,v))

• C14 = Resp*(*C2*,*C3) =—P(x,e,w) *v —P(x,1,u) v P(u,l*,w) *Cz = P*(s*,*e,*s*), C14 =—*P*(x, e,w) v —*P(x,l,u) v P(u,l,* w)

*0g* = [X +- S,W+s]*=mgu(P(s,e*,s*), P*(x*,e*,w))

• C15 = Res? (C3,C14*)=—P(s,l,u) v P(u,1,s) C*3 *= P(t,a,b), C*5 =—P*(s,1,u) v P(u,1,s)*

*0g* = [

s *t,lt auf b]= mgu(P(t,a,b), P(s,l,u))*

• C16 = Res. (C13,C15*)=P(b,a,l)*

*Cy=—P(b,a,t), C16 = P(b, a,t)*

• *C1* = Res(C16*,*C,)=0, so the set S is inconsistent and

*U1,U2,U3,U4,U*5 H*UG*.

We conclude that the statement *"If every element of a group G is its own inverse, then G is an Abelian group*" is valid. The resolution process is represented graphically by the following binary tree.

174

A Computational Approach to Classical Logics and Circuits

**Example 6**.11. Formalization of mathematical reasoning Prove the statement: *“There exists an infinite number of primes”.*

We do not have hypotheses, we need mathematical axioms in order to prove the statement. The axioms in mathematical language, predicate logic language and their clausal forms are provided below.

**W*I*US**

*a*j: xxx;

(Vx)*-le*ss(x,x),

*C = -less*(x,x) *az*: if x < y then y *\**x

(Wx)(Wy)(*le*ss(x, y) +-*les*s(y,x), C2 = -*le*ss(x, y) v-*le*ss(y,x) *a*z : x divides x

(Vx*)divides*(x,x),

*Cz = divides(*x,x) *04*: if x divides y and y divides z, then x divides z

(Vx)(Vy)(W2*)(divides(*x, y) A *divides*(y,z) *► divides(x*,z)),

C4 = *-divide*s(x, y) *v divides*(y,z) *v divides*(x,z)) *0*4 :if x < y then y does not divide x

(Vx)(Vy*)(less*(x,y) →*-divides(*y,x)), Cs=*-les*s(x, y) V*-divides*(y,x) *ag*: if y divides *F*(x)=x!+1, then x < y

(Vx)(Vy)(*divides(y, F(*x)) → *less*(x,y)) Co = *-divides(y, F*(x)*) v less*(x, y)

175

Reasoning modeling and program verification

*az :x <F*(x)

(*Vx)less(x, F(*x)*),*

*C7 = less(x, F*(x)) *ag*: if x is not prime, then there is a y such that y divides x, y is prime and y is less than x.

*U=*(Vx)(*-prime(*x) → (3y*)(divides(*y,x*)^ prime*(*y) « less*(y, x))) The prenex normal form of *U* is: *UP* = (Vx)(y)*(prime*(x*) v (divides*(y,x*)^ prime*(y*) less*(y,x))) The Skolem normal form without quantifiers of *U* is: *US9 = prime(x) v (divides(H(*x),x*) A prime(H(*x)*) A less(H*(x), x))) The clauses provided: *Cg = prime(*x) *v divides(H(x*),x), *H(*x) is a Skolem function *Co = prime(*x) *v prime(H(*x)) *C*ho *= prime(*x*) v less(H(*x), x))

The conclusion: “If x is a prime then there is a y such that y is a prime, x is less than y and *F(*x) is not less than y."

*C* = (Vx)*(prime(*x)-> (3y)*(prime(y*) *^ les*s(x, y) A *less(F*(x), y))) -C= (Vx*)(prime(*x) → (Ey)*(prime(y) Ales*s(x, y) A *less(F*(x), y))) =

= (3x)*(prime*(x) ^ (Vy*Gprine(*y) *v Gl*es*s*(x, y) v *less(F(*x), y))) = = (3x)(Vy*)(prime(x)^(prime(*y) *V-le*ss(x, y) *v less(F*(x), y)))

– prenex normal form The clausal normal form of -C is: *prime(a) ^ (prime*(y) *v l*ess*(a, y) v less(F(a)*, y)) providing the clauses: Ci*= prime(a),* C12 *= - prime(y*) *V-less(a, y) v less(F(a*), y*)*

We want to prove that the conclusion is derivable from the set of axioms:

*(1,2,az, 24, 25, 26, 27, ag.* Using resolution we prove that S=*{C*1,*C2*,...*,C*12} is an inconsistent set.

C12 = *-less(a*, y*) v less(F(a)*, y) *v prime*(y), *Cg = prime*(*x) v prime( H(x))*, C13 = Res Pux-x)(C9,612)=*-less(a, x) v less(F(a)*,x) *V[-prime*(x)]*v prime(H(*x)*) Cg = prime(x) v divides(H(*x),x),

*C12 =mprime*(y) v-*les*s(*a, y) v less(F(a)*, *y)*)

• C14 = Res Pure-xj(C3*, C*12)*=divides(H(*x),x) *V-less(a,x) v less(F*(a),x)

17*6*

A Computational Approach to Classical Logics and Circuits

•

C13 *= prime(H(*x)*) v-less(a,x) v less(F(a*), x) C12 =*-prime(y*) *V-less*(a, *y) v less(F(a)*, y) Cis = RespxH(x)]*(C*13,612)*=-less(a,x)v-less(a, H(x)) v less(F(a*), x*)*

*v less(F(a), H(*x))

*Co=-divides(u, F*(*v)) v less(v,u)* C16 = Respora,u*t-H*(x)(C6, C5)=*-less(a,*x*) v less(F(a*),x*) v less(F(a), H(x)*)

*V-divides(H(x), F(a*)) *C*s =-*le*ss(*u,* v) *v-divides*(v*,u)* C17 = Respué-F(a),*v+-H(x*)]*(C*5,C16)=*-less(a,*x) *v less( F(a*), x)

*v-divides(H(x),F(a)) C14 = divides{H*(x), x*)v-less*{*a, x) v less(F(a*),x) C18 = Restaj*(C*17*,C*14)*=-less(a, F(a))v less(F(a), F(a)) C7 = l*ess(x*, F*(x)) Cg = Res(C,*C18) = less(F(a), F(a))* G = *-les*s(x,x) C20 = Resfr-F(a)*(*C,C19)=0

•

•

•

•

The empty clause was derived from the set of clauses S={C1,*C5, C6,C2,C3,C9,* C12}, so there is a contradiction among the formulas *{U,,U2,...,U*3, -C}

By applying the “reductio ad absurdum' principle we conclude that C is derivable from the set of axioms and the statement:

*“There exists an infinite number of primes”* is true.

*1*77

Reasoning modeling and program verification

6.*3. Program verification using predicate logic*

We present an approach of the program verification task introduced in paper [62]. A program is described by a set of predicate formulas and then using the resolution method, the execution of the program is modeled. Let *p* be a *program*: an algorithm described using a chart flow, pseudocode or a programming language. For a program we have to solve these problems:

the *stop problem:* having an input data x, will the program stop? the *answer problem:* given the input data x which is the output data z of a

program?

• the *correctness problem:* if a program stops, will the relations input data –

output data be satisfied? the *equivalence problem:* having two programs, will they provide the same

output data for the same input data? A program uses three data vectors denoted by:

• x=(x1,...,x,) - the input vector (the variables are used only on the right side

of an assignment)

y=(y1...,xk) - the work vector

• z=(21,..., zm) - the output vector (the variables are used only on the left side of

an assignment)

**Definition 6.1.** To a program *P* we associate a graph G=(X*,U*) as follows: 1. there is a unique initial vertex (without incoming arcs) called *Start,* denoted by

S;

2. there is a unique final vertex (without outgoing arcs) called *Halt, d*enoted by

*H;*

3. each arc a that is not incident with *H* has associated a formula *Pa*(x, y) and

an assignment y = *fa*(x,y) 4. each arc a that is incident with *H* has associated a formula P. (x,y) and an

assignment z *= fa*(x,y) 5. if *a*; and *a*z are two outgoing arcs from the same vertex v *#H*, only one of

the formulas Pa, (x, y) and Paz (x,y) is true.

178

A Computational Approach to Classical Logics and Circuits

Note that for a given input data x, the vector y is well defined and the formulas

*P.(*x, y) can be evaluated as true or false for each arc *a.*

**Definition 6.2.** The *condition for accessing the node (vertex)* y, from S is expressed by the predicate *q;*(x, y) which must be true. Note: 95(x, y) =*T,* VX,Y. **Definition 6.3.** The *stop condition* is the condition for accessing *H* from S and it is expressed by the predicate *q*H(x,z), which must be true.

**Definition 6.4.** The formula W, is called th*e description formula for the arc a* =(Vi, v,) and has the form: Wą:9*;*(x,y) ^ *Pn*(x,y) →*q;*(x*, fa*(x, y)).

Note that We must be true if the arc *a* is traversed. This formula is in fact a clause: Wa=-9;(x,y) v-*Pn*(x,y) *v q;*(x*, fa*(x, y)). **Definition 6.5.** Let aj*...*.*, A,* be all the arcs of the graph associated to the program P. The formula *Up =* W., 1... *VW*. is called the *description formula of P*. We will also identify *U*p with the set of all description clauses of *P.*

**Theorem 6.1.** Let *P* be a program and *U*p the set of all description clauses of *P. Up* is a consistent set of clauses.

A program is characterized not only by *Up* but also by *U,* (a formula that defines the properties of input data) and U, (the set of formulas representing the axioms for the predicates Pa(x, y) and the functions f*a*(x,y)). **Definition 6.6.** *A program P is convergent* for an input data x if there is a path from S to *H* using X. *A program P is convergent* if for all x there is a path from S to *H.*

179

Reasoning modeling and program verification

**Theorem 6.2.** Let *P* be a program and *U,* a set of formulas obtained from *Up* by eliminating all the literals containing the predicate *qu*(x,z). *P* is convergent if and only if the set *U, UU, UU,* is inconsistent. **Definition** 6*.7.* A clause containing only the literal *qu*(x,z) is called a *halt clause.*

**Theorem 6.3.** Let *P* be a program and X the set of clauses corresponding to *UpUU, VU.*. The program P is convergent if and only if there is a deduction of a halt clause from X using resolution.

In the following we will study the convergence of the following program according to the theory presented above. Predicate resolution is used to model the execution of the program.

*Program P:* **input**: x = (x1,x2*,n)*

**output:**

*1*:2) or z:=*n*\*x2 (if *n72)* **begin**

y:=1; *i*:=0); if *(n*:2) **then**

while *(i<n)*

y:= y\* x];

*i:=i*+1; **end\_while else**

y:=*n*\* x2; **end\_if**

z:= y;

**end**

•

•

•

the input vector: x = (x1,x2*,n*); the output vector: z =(z); the work vector: y =*(*i, y)

180

A Computational Approach to Classical Logics and Circuits

We associate the following graph to the program *P*: 0

(S) Start vertex

*P = T* di i = 0, y=1

*Pas = div(n.*2) Ó'

a*z P*az *= -div(n,2)*

*z=1*\* x2

*Pas = less(i,n).*

*i=*i+1, y = y\* x1

*Pa =*

*less(i,n*)

2= y

Halt vertex

The description formulas for the arcs are as follows: Wa*, :98*(x1,x*2,n,i,y*) ^ P*a,* → 91(x1, x*2,8*,0,1) =*T AT91*(x1, x*2,*,0,1) =

*=91*(x1,x*2,11*,0,1) *Wa,* :*9*1(x1, x2,*n,i, y) A div(n,2*)+*92*(x1,x*2,n,i,y*) Wag :9*}*(x1,x2*,n,i,y) 4-div(1,2)→ 9H* (X*1, X2,1,n*\* xz) W *, :92*(X1, X*2,n, i, y) « less(i,n*)+*92*(X1, X*2, n,i*+1, y\* xi) W*as : 9*2(x1, x*2, n, i, y) ^-less(i,n) →H*(x1,x*2,*n, y)

Tel

92 (132*.1,* Tgl

11

The clausal normal forms of these formulas are: *C:*91(-41,4*2,1*1,0,1) *C2:*491(x1,x*2,n,i,* y) *V-div(n,2) v 9*2(x1, #*2,n,i, y) C*z: -*9*,(x1,x2*,n,i, y*) *v divín,2)vqH*(x1, x2,*n,n*\* x2) C4:7*9*2(x1,x*2,n,i,y*) v *less(i,n) v 92*(x1, x*2,n,i*+1, y\* xi). *C*s:-*9*2(x1,x*2,n,i, y) v less(i,n) v9H* (x1,x*2,n*, y)

*Up* = {C1,C2,C3,C4,C5} The axioms which define the predicates: *equal, les*s and *di*v for integers are expressed by predicate formulas having universal quantified variables. We omit the quantifiers obtaining the free formulas presented below.

The functions: *pred*(y) = y-1, *succ*(y) = y +1 are used.

**181**

Reasoning modeling and program verification

The binary predicate *equal* is defined by the axioms: *ei: equal(*x,x) **(reflexivity)** *ez: equal(x*, y) *→ equal(*y,x)=*-equal(x*, y) *v equal(y*,x) **(symmetry) ez: (transitivity)** *equal*(x, y) *equal(y,*z) *► equal*(x*,* z) = *equal(x,* y) *v Lequal(y,*z) *v equal(*x*,z*) *24: equal*(x,y) *→ equal(succ*(x), *succ(*y))= *nequal(*x*, y*) *v equal(succ*(x)*, succ(*y)) *eg: equal(x,* y) *→ equal(pred(*x*), pred*(y))= m*equal*(x, y) *v equal(pred(*x)*, pred*(y))

The binary predicate *le*ss is defined by the axioms: 1*0: l*ess(x,x) *11: l*ess(x,y) → *le*ss(y,x)=*-l*ess(x, y) V*-le*ss(y,x) *Iz: less(*0,1) *13: les*s(x, y) *→ l*ess(x, *succ(*y))= *le*ss(x, y) *v les*s(x, *succ*(y)) *14: less*(x, y) *→ less(succ(*x)*, succ*(y))=*-l*ess(x, y) *v less(succ*(x*), succ*(y)) *ls: le*ss(x,y) *→ less(pred*(x), y) = *l*ess(x, y) *v less(pred(*x), y) *16: l*ess(x,y) *→ less(pred(x), pred*(y))= *-l*ess(x, y*) v less(pred(x), pred(*y))

The predicate *div (divisibility by* 2) is defined by the axioms: *di: div*(0,2) *dz: div(k,2*) *► div(k* +2,2)=*-div(k,2) v div(k +*2,2) *dz: div(k,*2) *► diví*k -2,2)= *mdiv(k,2) v divík* – 2,2) *d4: div(k,*2)*-divík* +1,2)= *divík*,2)*v-diví*k +1,2) *ds: divík,2*) +-*divík* – 1,2)=*-divík,*2)*v-divík* – 1,2)

The set of formulas representing the axioms for the predicates and the functions used in the description of the program is

*Un* = {(1,*22,2*3,2*4,25,lo,k,12,13,14,15,16, dq,d2,d3,d4,d5}, U,* = 0

For *n*=3 *(n72)* we prove the convergence of *P*. We follow the resolution process in order to derive a halt clause and obtain the output data. At each resolution step the corresponding unifier is written.

Co = Resfit 0,57-1*(C,*C3*)=di*v(3*,2) v qu* (x1, x2,3,3\* x2) *do*'= Resf-oj*[dı,dz)= div(*2,2) *dy*'= Res xx-2*)(do',da)=-d*iv(3,2) *C* = Res(*C6,07')=9H* (X1, X2,3,3 \* x2) halt clause, the output data is z = 3 \* Xy *UpUU*, Fres Cy, therefor*e the program P is convergent.*

182

A Computational Approach to Classical Logics and Circuits

For *n*=4 (*1:2*) we prove the convergence of *P*. In the while loop four iterations are executed and the resolution process models this execution as follows:

IS.

Crewl1*)*

*C*g = Res Pik 0,461,*(*C,C2)=*div(*4*,2) v 92*(x1, 42,4,0,1) *do*'=Resfkoj*[dı,dz)= di*v*(2,2) do*'= Resikr-*2(d6',d2) = di*v(4,2) *c*ọ = Res*(*Ce*,de')* = *92*(x1,x2,4,0,1)

Cpo = Resfix0,-1)(C9,C4*)=-le*ss(0,4) *v 92*(x1,x2,4,1,xj) Using the axioms lo.*..., ko* we derive *ly'= le*ss(0,4) as follows: G = Res(*12,13) = less(*0,2), Cz = Res(9*,13) = less(*0,3) *ly'*= Res(c*3,13) = le*ss*(*0,4)

**Iteration 1:** Co = Res(Cio*l;') = 92*(x1, x2,4,1, xı) Cio = Restit), ytx,1(C),C4)=*-le*ss(1,4) v *9*2(x1, x2,4,2, x1 \* xj) Using the axioms *lo....lo* we derive l*g'= le*ss(1,4)

**Iteration 2:** C} = Res(*Co,*lz*') = 9*2(x1,x2,4,2, x2 \* x;) *C*o = Respir-2,34-4,\*x,]*(*C3,C4)=*-less*(2,4) v *92*(x1,x2,4,3,x2 \* x2 \* x) Using the axioms l*os...el6 w*e derive *ly'= less(2,*4)

Iteration 3: Ci = Res(Co*,l9*') = *92* (x1,x2,4,3,x; \* x \* x) *C*o = Resfík–3,94–3; \*\*x,](C$*,*Ca)=-*le*ss(3,4) v *92*(+],x2,4,4,xj \* xj \* x2 \* xy) Using the axioms *lo.*.*..lo* we derive l*i'= les*s(3,4)

**Iteration 4:** cm = Res(*Co, 40')=92*(X1, X2,4,4,x1 \* x2 \* x2 \* xj) The exit of the loop: Cuj = Res(C4*,C*5*) = le*ss(4,4) *v9*h(x), X2,4, x1 \* x1 \* x2 \* xi) l*o =-le*ss(4,4),[x+ 4] *C*12 = Res(C11*,6)= 9h* (x1, x2,4, x1 \* x2 \* x1 \* xj) *C*12 is a halt clause and provides the output data: 2 = x2 \* x1 \* x1 \* x1 *UpUU,* Fres C12, therefor*e P is a convergent program.*

183

Boolean Algebras and Boolean Functions

IS

*7.* **BOOLEAN ALGEBRAS AND BOOLEAN FUNCTIONS**

George Boole introduced Boolean algebras in 1854 in the paper [7]. In 1938 Claude Shannon proved that a two-valued binary Boolean algebra can describe the operations of two-valued electrical switching circuits. In modern times Boolean algebras and Boolean functions are indispensable in the design of computer chips and digital circuits.

In this chapter the theoretical concepts of Boolean algebras and Boolean functions are introduced. These concepts are used further in the simplification of Boolean functions. Three simplification methods: Veitch-Karnaugh diagrams method, Quine's method and Moisil's method are described and applied in numerous examples. The following papers (8, 14, 25, 26, 37, 53, 62] were used as bibliographic references.

nimit

*7.1. Boolean algebras* **Definition** 7*.*1. **A Boolean algebr**a is a structure *(A*, ^, *V*, 50, 1) described by the following axioms:

1. |A|22, A contains at least two elements: 0 and 1, 0+1 2. ^, V are binary operations

is a unary operation 4. O is the zero element:

XA0 = 0 1 x = 0 and xv0-0v x= x, Vx E *A* 5. I is the unity element:

XA1=1^x=x and xv1=1 v x = 1, Vxe A 6. O is the first element, 1 is the last element

x A x = 0 and x V x = 1, VXE A

7. double negation: x = x, Vxe *A* 8. commutativity:

x^ y = y^x and xv y = y v x,Vx, y E A 9. associativity:

x^(*y*^2)=(x^ y) 42 (= XA YA Z) and

xv(y v z)=(xv y) v z (=x*v y*vz),Vx,y,z e A 10. distributivity:

x^(y v z) = (x^ y) v(x A z) and xv(y^2)=(xv y) ^ (XV z), Vx, y, z E *A*

**184**

A Computational Approach to Classical Logics and Circuits

11. idempotency:

x^x= x and xv x= x, VXE AVEA 12. De Morgan's laws:

x^ y=xv y and xv y = x^ y, Vx, y E A 13. absorption laws:

x^(xv y) = x and xv(x^ y) = x, Vx, ye A

**Note: A Boolean algebra** is a complemented distributive lattice. In the literature there are alternative symbols used for the binary operations of a Boolean algebra: (1,V) or (\*,+). In this paper we use the pair of symbols: (^,v).

The d**uality principl**e in a Boolean algebra: „For any equality of two Boolean expressions, *U =*V, there is another equality, *U'V*', obtained by interchanging the Boolean operations: A, v and the Boolean values: 0,1".

**Example 7.1.** The **binary Boolean algebr**a is *B = (B2* ={0,1}, ^, *V*, , 0, 1), where the truth tables for the Boolean operations are as follows:

0

1 1

1

0

1

1.

O

**Example** 7*.2.* The structure *(FP, 1, V*, 7*, F, T*) is a Boolean algebra.

*• F*p is the set of all well-formed propositional formulas;

• binary operations: A (conjunction), v (disjunction);

unary operation: (negation); zero element: *F*, unity element: *T*, where *F(false), T(true)* are the truth values.

Propositional logic is a logical system that is intimately connected to Boolean algebra. Many syntactic concepts of Boolean algebra carry over to propositional logic with only minor changes in notation and terminology, while the semantics of propositional logic are defined via Boolean algebras in a way that the tautologies (theorems) of propositional logic correspond to equational theorems of Boolean algebra.

The semantics of propositional logic is based on truth assignments. The basic idea of a truth assignment is that the propositional variables are mapped to elements of a fixed Boolean algebra, and then the truth value of a propositional formula is the element of the Boolean algebra that is obtained by computing the value of the Boolean expression corresponding to the formula.

**185**

Boolean Algebras and Boolean Functions

**Examp**le 7.3. The structure *(P(*X), n, U, *C*y, 0, X) is a Boolean algebra.

• X is a set and P*(*X) is the set of all subsets of X;

binary operations: n (intersection), u (reunion); unary operation: Cx(A)= X\A - complementary set of A with respect to X ;

zero element: Ø and unity element: X. The properties from Definition 7.1 can be easily verified.

7*.2. Boolean functions* Definition *7.2.* Let *B =(B2, 1, V*, 7,0,1) be the binary Boolean algebra, *B2* ={0,1} and *neN.* A *Boolean function of n variables* is a function *f :(B2*)" → B2 defined as follows: 1. the projection function: *P,: B” → B2, P,* (\*..., X; ..., x,y) = x;, is a Boolean

function. 2. if *f,g:B" - B2* are Boolean functions then *f 18, fvg, f* are Boolean

functions of *n* variables, where: *(f^*g)(x1,..., X*,) = f (*x1,..., Xy) ^ g(xJy..., X); *(f v g*)(xpo..., xn) *= f* (x1 ..., xn) v g(x2,..., Xin);

*f* (x2...., Xn) = f(x)....,xn). 3. any Boolean function can be obtained by the application of the rules 1 and 2. Theorem *7*.1. *Vn*en\*, there exist 22" Boolean functions of n variables.

Theorem *7.*2. The structure (FB(n), ^, *v*, *; fo, f«z* , ) is a Boolean algebra, where FB*(*n) is the set of all Boolean functions of neN\* variables, where A, V and are defined in the above definition and *f*o(X1,..., X,) = 0 and *fro*zu , (xp ..., Xn)=1 are the constant functions corresponding to *contradiction* an*d tautology* respectively. **Example** 7.4. For n= 1, there exist 24 = 4 Boolean functions of one variable represented by their expressions and tables of (truth) values as follows:

*fo*(x)=0 | *contradiction*

*tautology* | 0

0

0

1

1 L 1 1 0 1 1 1 0

1

*f*i(x)= x '

f(x) =ř

*f*o(x)=1

186

A Computational Approach to Classical Logics and Circuits

Example 7.5. The 2 = 24 = 16 Boolean functions of 2 variables are represented in the table below. xy fo(x, y) fi(x,y) 12(x,y) | fz(x,y) f4(x, y) 15(x, y) | f6(x, y) | f7(x,y)

*contra*

*die*n | x Ay | xay | x | Xay 1 y | xe y | xvy 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0

1 1 1 110 0 0 1 1 0 1 0 1 1 11 0 1 1 0 1 1 0 1 1 0

x y fo(x, y) 19(x,y) 110(x,y) fu1(x,y) 512(x,y) [13(x, y) | 514(x,y) | *f1*5(x,y)

xty |

*tauto* x +y.

y → x

x →yx Tv

*logy* 0011111111111 10 1 0 0 0 0 1 1 1 1

110 0 1 0 1 1 1

0 0 1 1 1111 0

0 1 0 1 0 1 1

Each function has an inverse function obtained as its negation. The pairs of inverse functions are: *(fo,fi5), (*f1, f14*), ($2*,f*13), (13,* f*12), (f4*, f*u), (fs,*f*io), ($6,fg), (67,fg).* We use the following symbols for operations:

• (nor) – Pierce's function, x y=(xv y) = x^y

• 1 (nand) – Schaffer's function, x T y = (x ^ y)=xvy

• → (logical implication), x+y=xv y

• \* (logical equivalence), x + y =(x y)^(y + x) = (x y) v(x^y)

• („or”- exclusive ‘or”), xəy=(x + y) = (\* ^ y) V (x ^y)

**Y**

For a uniform representation of the variables and their negations we introduce

if a=1 the following notation: xa

\* Le if a=0? For x, a e{0, 1} we have: x° = x, x'=x and

0° = 7=1; O'=0; 1° = 1 = 0; 1' =1 *ra* - Ji, if x= a

if x # *a'*

X, a €{0, 1}.

T

187

Boolean Algebras and Boolean Functions

7.*3. Canonical forms of Boolean functions*

All Boolean expressions, regardless of their form, can be transformed into two standard (canonical) forms: the *disjunction-of-conjunctions* form or the *conjunction-of-disjunctions* form.

In the (\*, +) notation of the Boolean operations, these forms are called *sum-of-products (SOP*) and *product-of-sums (PO*S) respectively. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

**Th**eorem 7.3. A Boolean function *f :(B2*)" — *B2,ne*N\* can be transformed into two equivalent forms: 1*. disjunctive canonical form (DCF): disjunction* of conjunctions

(1) *f*(x1.... x,y) = *V (f (*p..., an)^x;^! A.... A X,9")

(ap,..., *QVEB 2. conjunctive canonical form (CCF): conjunction* of disjunctions (2*) f*(x1..., xn) = A *(f(Q*1 ...*, Q*.*,)* v xj^! V... Vx,C")

(Q1,...*, , )EB*"

Theorem 7.4. A Boolean function *f :(B2)" → B*2 is unique determined by its values *f(a*,..., An), where (C1,..., *An) eB:* 1. *disjunctive canonical form (DCF):*

(1) 6 (l') f(x{..., Xn) =

(x," A... AXN

(Q1,...,*Q,*)*EB*" and f(*aga*, ,)=1 2*. conjunctive canonical form (CCF):*

(2) A (2*') f*(x1,..., xn) =

(*Q), Q,*D*EB*" and *f(a*,.*..,* ,,)=0

*(*x,ai V... *V* Xn")

Example 7.6. For n=2, the disjunctive canonical forms of the functions f*3,613, 56,* f1*2* from Example 7.5 are built using formula (1):

fg(x, y)=(1^° ^yos von xony') von x'ny Osv(onx' ^y')=zny $13(x, y) = (Inxo yo) v (14 x° ^y') vonx' nyo) v (1x'y') =

= (x^y) *v*(x^ y)v(xy) fo(x, y) = (01 x° nyo)v (Inxony')v(Inx'ny)vconx' ^y') = (any) v(xny) $*12*(x,y)=(1^° ^yov (14 x° ^y')vonx'nyo) v (onx' ~ y') =

=(x^y) v(an y)=x^(av y)= x

188