Eric Wan – ezw23@drexel.edu

Part 1:
2017/12/04 - 19:00 – 23:00
2017/12/05 – 16:45 – 20:00
2017/12/06 – 14:00 – 20:00
2017/12/07 – 15:00 – 20:00
2017/12/08 – 16:00 – 23:00

Difficulties:

- Everything, I am still unsure of what is happening with the proofs other than it runs through the code to try to prove equality.

Part 2:
(and (is-cnf? expr1) (is-cnf? expr2)) -> (is-cnf? (list ‘and expr1 expr2)))

Assume (and (is-cnf? expr1) (is-cnf? expr2)) = #t
Justify the right side:
(is-dnf? (list ‘and expr1 expr2))
= (and (nnf? expr) (no-or-above-and? expr))

(nnf? expr)
= (conjunction? expr)
= (and (nnf? (op1 expr)) (nnf? (op2 expr)))
= (and (nnf? expr1) (nnf? expr2))
by assumption = (and #t #t) = #t

(no-or-above-and? expr)
= (conjunction? expr)
= (and (no-or-above-and? (op1 expr)) (no-or-above-and? (op2 expr)))
= (and (no-or-above-and? expr1) (no-or-above-and? expr2))
by assumption = (and #t #t) = #t

(is-cnf? (list ‘and expr1 expr2))
= (and (nnf? expr) (no-or-above-and? expr))
by smaller proofs 1 & 2
= (and #t #t) = #t

Part 3:
(and (is-cnf? e1) (is-cnf? e2)) → (is-cnf? (distrib-orand e1 e2))

looking at def:

```
(define (distrib-orand expr1 expr2)
  (cond
    [(conjunction? expr1) (list 'and (distrib-orand (op1 expr1) expr2)
                                (distrib-orand (op2 expr1) expr2)) ]
    [(conjunction? expr2) (list 'and (distrib-orand expr1 (op1 expr2))
                                (distrib-orand expr1 (op2 expr2))) ]
    [ else (list 'or expr1 expr2) ]
  )
)
```

(distrib-orand x y)
= (or x y)

(distrib-orand (and x y) z)
= (and (distrib x z) (distrib y z))
= (and (or x z) (or y z))

(distrib-orand x (and y z))
= (and (distrib x y) (distrib x z))
= (and (or x y) (or x z))

Original question:
(and (is-cnf? e1) (is-cnf? e2)) → (is-cnf? (distrib-orand e1 e2))

Base Case:
Let e1 be an expression that DOES NOT start with AND
Let e2 be an expression that DOES NOT start with AND
Let (and (is-cnf? e1) (is-cnf? e2)) = #t

Proof:
(is-cnf? (distrib-orand e1 e2))
= (is-cnf? (or e1 e2))
= (and (nnf? expr) (no-or-above-and? expr))

(nnf? expr)
= (disjunction? expr)
= (and (nnf? (op1 expr)) (nnf? (op2 expr)))
= (and (nnf? e1) (nnf? e2))
by assumption = (and #t #t)

(no-or-above-and? expr)
= (dusjunction? expr)
= (and (no-and? (op1 expr)) (no-and? (op2 expr)))
= (and (no-and? e1) (no-and? e2))
by assumption = (and #t #t) = #t

(is-cnf? (distrib-orand e1 e2))
= (and (nnf? expr) (no-or-above-and? expr))
= (and #t #t) = #t

Inductive Case 1:
(distrib-orand (and x y) z) = (and (distrib-orand x z) (distrib-orand y z))

Hypothesis:
Assume: e1 = (and x y), e2 = z
(is-cnf (and x y)) = #t
(is-nnf? x) = #t
(is-nnf? y) = #t
(no-or-above-and? x) = #t
(no-or-above-and? y) = #t
(is-cnf? z) = #t
(is-nnf? z) = #t
(no-or-above-and? z) = #t
Assume:
(is-cnf? (distrib-orand x z)) = #t
(is-cnf? (distrib-orand y z)) = #t

(is-cnf? (distrib-orand (and x y) z))
By def of distrib-orand:
= (is-cnf? (and (distrib-orand x z) (distrib-orand y z)))
Two cases
(is-nff? (or (distrib-orand x z) (distrib-orand y z)))
= (and (is-nff? (distrib-orand x z)) (is-nnf? (distrib-orand y z)))
= (and #t #t) = #t
(no-or-above-and? (or (distrib-orand x z) (distrib-orand y z)))
= (and (no-and? (or x z)) (no-and? (or y z))
= (and #t #t) = #t
Thus
(is-cnf? (and (distrib-orand x z) (distrib-orand y z)))
= (and #t #t) = #t

Inductive Case 2:
(distrib-orand x (and y z)) = (and (distrib-orand x y) (distrib-orand x z))

Hypothesis:
Assume: e1 = x, e2 = (and y z)
(is-cnf x) = #t
(is-nnf? x) = #t
(no-or-above-and? x) = #t
(is-cnf? (and y z)) = #t
(is-nnf? y) = #t

(is-nnf? z) = #t
(no-or-above-and? y) = #t
(no-or-above-and? z) = #t
Assume:
(is-cnf? (distrib-orand x y)) = #t
(is-cnf? (distrib-orand x z)) = #t

(is-cnf? (distrib-orand x (and y z)))
By def of distrib-orand:
= (is-cnf? (and (distrib-orand x y) (distrib-orand x z)))
Two cases
(is-nff? (or (distrib-orand x y) (distrib-orand x z)))
= (and (is-nff? (distrib-orand x y)) (is-nnf? (distrib-orand x z)))
= (and #t #t) = #t
(no-or-above-and? (or (distrib-orand x y) (distrib-orand (x z)))
= (and (no-and? (or x y)) (no-and? (or x z))
= (and #t #t) = #t
Thus
(is-cnf? (and (distrib-orand x y) (distrib-orand x z)))
= (and #t #t) = #t

Part 4:
(bool-eval (list ‘or expr1 expr2) env) = (bool-eval (distrib-orand expr1 expr2) env)
(list ‘or expr1 expr2) = (distrib-orand expr1 expr2)

Assume (conjunction? expr1) = #t
=> (and E1 E2)
(distrib-orand expr1 expr2)
b/c expr1 is a conjunction:
= (list ‘and (distrib-orand (op1 expr1) expr2) (distrib-orand (op2 expr1) expr2)
b/c expr1 is a conjunction:
= (list ‘and (list ‘or (op1 expr1) expr2) (list ‘or (op2 expr1) expr2))
Assuming the parts expr1 are not conjunctions
= (and (or (op1 expr1) expr2) (or (op2 expr1) expr2))
= (and (or E1 expr2) (or E2 expr2))
(list ‘or expr1 expr2)
= (or expr1 expr2)
= (or (and E1 E2) expr2)

Assume (conjuction? expr2) = #t
=> (and E3 E4)
(distrib-orand expr1 expr2)
b/c expr2 is a conjunction
= (list ‘and (distrib-orand expr1 (op1 expr2)) (distrib-orand expr1 (op2 expr2)))

= (list ‘and (distrib-orand expr1 E3) (distrib-orand expr1 E4))
= (list ‘and (list ‘or expr1 E3) (list ‘or expr1 E4))
= (and (or expr1 E3) (or exp1 E4))
(list ‘or expr1 expr2)
= (or expr1 expr2)
= (or expr1 (and E3 E4))

Assume (and (not (conjunction? expr1)) (not (conjunction? expr2))) = #t
=> (not (conjunction? expr1)) = #t
=> (conjunction? expr1) = #f
=> (not (conjunction? expr2) = #t
=> (conjunction? expr2) = #f

(distrib-orand expr1 expr2)
Both expr1 and expr2 are not conjunctions
Assuming both expr1 and expr2 have conjunctions within their terms
(distrib-orand E1 E2)
= (list ‘or E1 E2)
= (or E1 E2)
(list ‘or expr1 expr2)
= (or E1 E2)

Part 5:
(is-cnf? (cnf expr)) = #t
1. Prove k=0,1 case is true
(is-cnf? (nnf2cnf (nnf expr)))
if expr is constant:
= (is-cnf? (nnf2cnf (constant)))
= (is-cnf? (constant? constant))
= (is-cnf? constant)
= (and (nnf? constant) (no-or-above-and? constant))
= (and (constant? constant) (constant? constant))
= (and #t #t) = #t
if expr is variable:
= (is-cnf? (nnf2cnf (variable)))
= (is-cnf? (variable? variable))
= (is-cnf? variable)
= (and (nnf? variable) (no-or-above-and? variable))
= (and (variable? variable) (variable? variable))
= (and #t #t) = #t
if expr is negation (not var):
= (is-cnf? (nnf2cnf (negation)))
= (is-cnf? (negation? negation))
= (is-cnf? negation)

= (and (nnf? negation) (no-or-above-and? negation))
= (and (variable? (op1 negation)) (no-or-above-and? negation))
= (and (variable? variable) (no-or-above-and? negation))
= (and #t #t) = #t

2. Assume k=n case is true
Assuming E1 and E2 are in cnf:

3. Prove k=n+1 case is true
if expr is conjunction (and-expr):
= (is-cnf? (nnf2cnf (and E1 E2)))
= (and (nnf? (nnf2cnf (and E1 E2))) (no-or-above-and? (nnf2cnf (and E1 E2))))
= (and (nnf? (list ‘and (cnf E1) (cnf E2))) (no-or-above-and? (list ‘and (cnf E1) (cnf E2))))
= (and (nnf? (list ‘and E1 E2)) (no-or-above-and? (list ‘and E1 E2)))
= (and (conjunction? (and E1 E2)) (no-or-above-and? (and E1 E2)))
= (and (and (nnf? E1) (nnf? E2)) (no-or-above-and? (and E1 E2)))
= (and (and #t #t) (conjunction? (and E1 E2)))
= (and #t (and (no-or-above-and? E1) (no-or-above-and? E2))
= (and #t (and #t #t)) = (and #t #t) = #t

if expr is disjunction (or-expr):
= (is-cnf? (nnf2cnf (or E1 E2)))
= (and (nnf? (nnf2cnf (or E1 E2))) (no-or-above-and? (nnf2cnf (or E1 E2))))
= (and (nnf? (disjunction? expr) (no-or-above-and? (nnf2cnf (or E1 E2))))
= (and (nnf? (distrib-orand (cnf E1) (cnf E2)) …)
= (and (nnf? (distrib-orand E1 E2) …)
= (and (nnf? (list 'or E1 E2)) (no-or-above-and? (nnf2cnf (or E1 E2))))
= (and (disjunction? expr) (no-or-above-and? (nnf2cnf (or E1 E2))))
= (and (and (nnf? E1) (nnf? E2)) …)
= (and (and #t #t) (no-or-above-and? (nnf2cnf (or E1 E2))))
= (and #t (no-or-above-and? (disjunction? expr)))
= (and #t (no-or-above-and? (distrib-orand (cnf E1) (cnf E2))))
= (and #t (no-or-above-and? (list 'or E1 E2)))
= (and #t (disjunction? (list 'or E1 E2)))
= (and #t (and (no-and? E1) (no-and? E2))
= (and #t (and #t #t)) = (and #t #t) = #t

Part 6:
(bool-eval (cnf expr) env) = (bool-eval expr env)
(cnf expr) = expr
1. Prove k=0,1 case is true
if expr is a constant (c):
(cnf c)
= (nnf2cnf (nnf c))

= (nnf2cnf (constant? c)
= (nnf2cnf c)
= (constant? c)
= c = expr

(cnf v)
= (nnf2cnf (nnf v))
= (nnf2cnf (variable? v)
= (nnf2cnf v)
= (variable? v)
= v = expr

if expr is a negation (n v):
(cnf (n v))
= (nnf2cnf (nnf (n v)))
= (nnf2cnf (negation? (n v)))
= (nnf2cnf (nnf-not (n v)))
= (nnf2cnf (variable? (op (n v))))
= (nnf2cnf (n v))
= (negation? (n v))
= (n v) = expr

2. Assume k=n case is true
Assume that (cnf expr) outputs expr as it is already in cnf
Assume that (cnf expr1) outputs expr as it is already in cnf
Assume that (cnf expr2) outputs expr as it is already in cnf

3. Prove k=n+1 case is true
if expr is a conjunction (and expr1 expr2):
(cnf (and expr1 expr2))
= (nnf2cnf (nnf (and expr1 expr2)))
= (nnf2cnf (conjunction? (and expr1 expr2)))
= (nnf2cnf (list 'and (nnf (op1 expr)) (nnf (op2 expr))))
= (nnf2cnf (list ‘and (nnf expr1) (nnf expr2))))
= (nnf2cnf (and expr1 expr2))
= (conjunction? (and expr1 expr2))
= (list 'and (cnf (op1 expr)) (cnf (op2 expr)))
= (list 'and (cnf expr1) (cnf expr2))
= (list and expr1 expr2)
= (and expr1 expr2) = expr

if expr is a disjunction:
if expr is a conjunction (or expr1 expr2):
(cnf (or expr1 expr2))
= (nnf2cnf (nnf (or expr1 expr2)))

```
= (nnf2cnf (disjunction? (or expr1 expr2)))
= (nnf2cnf (list 'or (nnf (op1 expr)) (nnf (op2 expr))))
= (nnf2cnf (list or (nnf expr1) (nnf expr2)))
= (nnf2cnf (list or expr1 expr2))
= (nnf2cnf (or expr1 expr2))
= (disjunction? (or expr1 expr2))
= (distrib-orand (cnf (op1 expr)) (cnf (op2 expr)))
= (distrib-orand (cnf expr1) (cnf expr2))
= (distrib-orand expr1 expr2)
= (list or expr1 expr2)
= (or expr1 expr2) = expr
```