Avishai Yaniv 307916023
Alex Abramov 314129438

Question 2

c. relatives(joseph, Y), female(Y).

Question 3

a. A = {mn(ttt,M,mn,V,HO,mn,br(M))}, B = {mn(ttt,br(HO),mn,br(M),br(V),mn,br(M))}

S = {ttt = ttt} // ignored

S = {M = br(HO)}

A◦S = {mn(ttt,br(HO),mn,V,HO,mn,br(br(HO)))}

B◦S = {mn(ttt,br(HO),mn,br(br(HO)),br(V),mn,br(br(HO)))}

S = {M = br(HO), mn = mn} // ignored

S = {M = br(HO), V = br(M)} => S = {M = br(HO), V = br(br(HO))}

A◦S = {mn(ttt,br(HO),mn,br(br(HO)),HO,mn,br(br(HO)))}

B◦S = {mn(ttt,br(HO),mn,br(br(HO)),br(br(br(HO))),mn,br(br(HO)))}

S = {M = br(HO), V = br(br(HO)), HO = br(br(br(HO)))}

Failure: HO = br(br(br(HO))) is a recursive calculation, thus cannot be evaluated.

b. A = {m(M,N)}, B = {n(M,N)}

S = {m(M,N) = n(M,N)}

Failure: m cannot be evaluated as n.

c. A = {kmr(pt(pt),V,pt,g,kmr(TF),pt)}

B = {kmr(pt(V),V,pt,g,kmr(TF),HO)}

S = {kmr(pt(pt) = kmr(pt(V)} => S = {pt = V}

A◦S = {kmr(V(V),V,V,g,kmr(TF),V)}

B◦S = {kmr(V(V),V,V,g,kmr(TF),HO)}

S = {pt = V, V = V, V = V, g = g, kmr(TF) = kmr(TF)} // ignored

S = {pt = V, V = HO} => S = {pt = HO, V = HO}

A◦S = {kmr(HO(HO),HO,HO,g,kmr(TF),HO)}

B◦S = {kmr(HO(HO),HO,HO,g,kmr(TF),HO)}

Answer: S = {pt = HO, V = HO}

d. A = {mn([T])}, B = {mn(T)}

S = {mn([T]) = mn(T)} => S = {[T] = T}

A◦S = {mn(T)(

Failure: T=[[[[[...[T]...]]]]], is recursive and cannot be assigned.

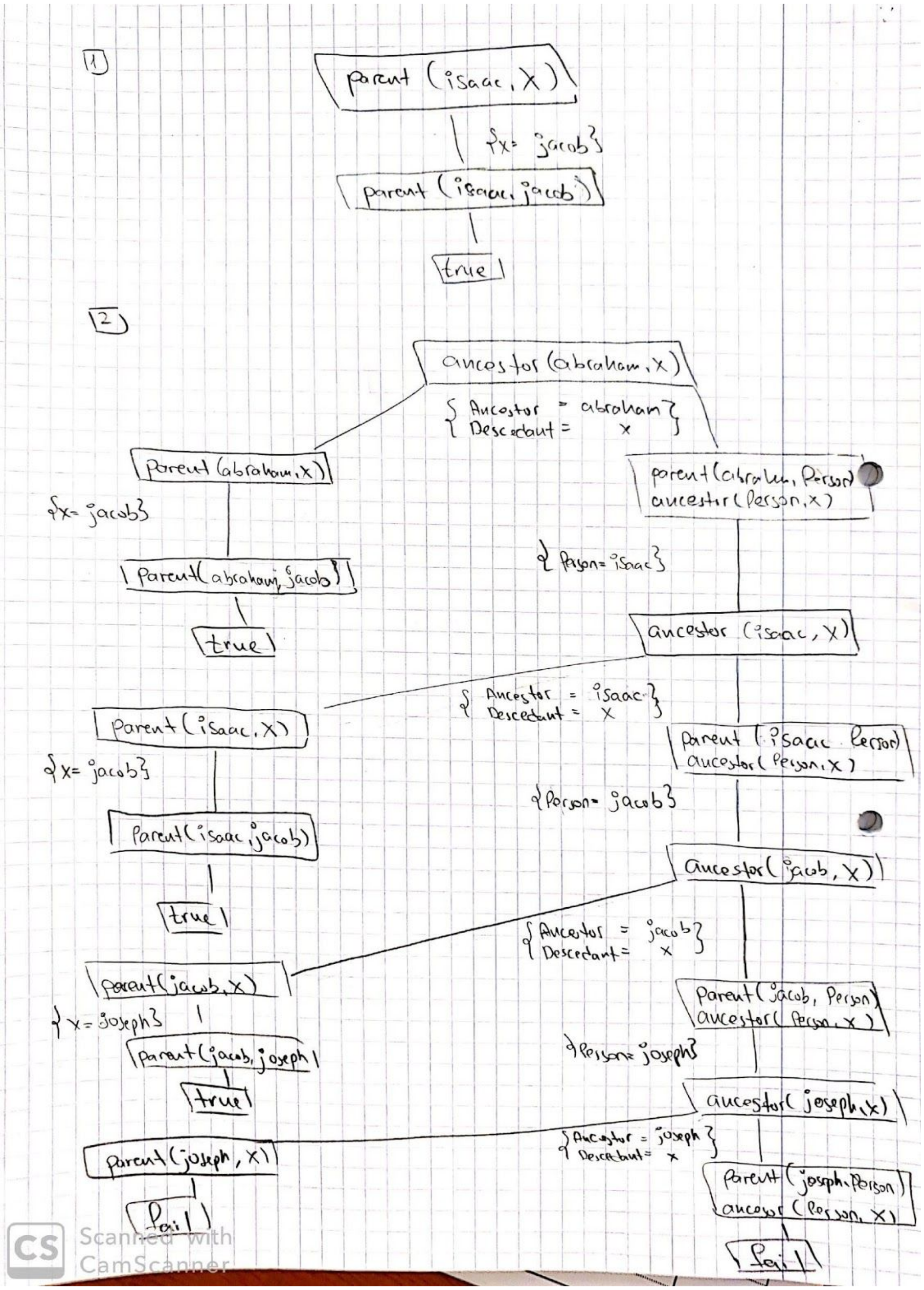
e. A = {g([br | [KK | K]])}, B = {g([[br | KK] | K])}

S = {g([br | [KK | K]]) = g([[br | KK] | K])} => S = {[br | [KK | K]] = [[br | KK] | K]} =>
S = {br = [br | KK], [KK | K] = K}

Failure: {br = [br | KK] is a recursive calculation, thus cannot be evaluated.

Question 4

a.



b. Answers of the answer-query algorithm:

1. X = Jacob
2. X = isaac
   X = jacob
   X = joseph

c. Both trees are finite because all paths included in the trees are finit, and are stopped at either ‘true’ node or ‘fail’ node.

d. Both trees are success trees because each one of them has a success path.

Question 5

a.

SC --> AtomC | CompoundC
AtomC --> NumberC | BooleanC | StringC
CompoundC --> RegularFormC | SpecialFormC
RegularFormC --> '(' StringC SC* ')'
SpecialFormC → ‘(‘ ‘lambda’ ‘(‘ StringC* ‘)’ SC* ‘)’ | ‘(‘ ‘(‘ ‘lambda’ ‘(‘ StringC* ‘)’ SC* ‘)’ SC*’)’
NumberC --> <number>
BooleanC --> #t | #f
StringC --> <string>