FP: 1. Seminarska naloga

Sebastian Mežnar (27192031)

Metode

eval:

Funkcija eval sprejme izraz in seznam vrednosti spremenljivk. Funkcija nato izračuna logično vrednost izraza, kjer spremenljivke zamenja z njihovimi vrednostmi. Če spremenljivka nima vrednosti, vržemo izjemo. To sem naredil z ujemanjem vzorcev. Pri tem se najprej evalvirajo potrebni podizrazi, nato pa se rezultat izračuna s pomočjo njihovih vrednosti.

removeEmpty:

Metoda sprejme izraz in vrne izraz, v katerem Or [] → False, Or [e] → e, And [] →True, And [e] →e. To sem naredil z ujemanjem vzorcev in pomočjo funkcije map. Izrazi se poenostavljajo rekurzivno.

removeConstants:

Metoda sprejme izraz in vrne izraz, v katerem odstranimo konstante. To sem naredil z ujemanjem vzorcev za vsak tip izraza posebaj. Pri izrazih tipa And in Or sem si pri tem pomagal s pomočjo funkcije foldr in pomožno funkcijo, ki sprejme element iz seznama in izraz ter vrne izraz. Če je izraz ki ga prejme False/True, to le pošlje naprej, čene vrne izraz tipa And/Or, ki je ustrezno spremenljen glede na rezultatu klica removeConstants na elementu iz seznama. Tukaj se najprej evalvirajo potrebni podizrazi in šele nato se izraz ustrezno predela.

pushNegations:

Metoda sprejme izraz in vrne izraz, v katerem so negacije porinjene v liste. To sem naredil s pomočjo ujemanja vzorcev. Izraze sem spreminjal od vrha drevesa proti listom, pri tem pa sem si v izrazih tipa And in Or pomagal z funkcijo map. V primeru Not(True/False) spremenim logično vrednost, saj je s tem izraz bolj poenostavljen.

removeVars:

Metoda sprejme izraz ter vrne izraz, v katerem se znebimo spremenljiv tam, kjer je to mogoče. To sem naredil z ujemanjem vzorcev že spremenjenih podizrazov za vsak tip izraza posebaj. V fukciji sem dodal tudi, da se izrazi tipa And in Or odstanijo, če so notranje spremenljivke v ustrezne (And [Variable 1, Not(Variable 1)] se poenostavi v False in podobno). Pri tem sem si pomagal s pomožnimi funkcijami rmDup, checkOne in checkAll. RmDup odstani iz seznama vrednosti, ki se večkrat pojavijo, checkOne preveri, v kakšem razmerju je trenutna vrednost z vrednostmi iz preostanka seznama in vrne ustrezno poenostavljen seznam. Funkcija checkAll gre čez vse elemente v seznamu in jih prilagodi glede na seznam, ki ga vrne funkcija checkOne in funkcija checkAll na preostanku seznama.

simplify:

Metoda sprejme izraz ter vrne poenostavljen izraz. Ta funkcija uporablja zgoraj omenjene funkcije in z njihovo pomočjo vhodni izraz poenostavi. Funkcije so klicane v zaporedju: pushNegations →removeVars → removeConstants → removeEmpty, saj menim, da je ta vrstni red najbolj ustrezen. Zaporedje funkcij se kliče

na vsakem vozlišču z začetkom v listih, saj lahko le tako zagotovimo, da se drevo do konca poenostavi. Čeprav se zaporedje funkcij velikokrat kliče, funkcija ni zelo časovno potratna (približno $O(n*\log_2 n)$).

tseytinTransformation:

Metoda sprejme seznam spremenljivk in izraz, ki ga spremeni v KNO s pomočjo Tseytin transformacije. Funkcijo sem implementiral iz več delov. Na dobljenem izrazu se uporabi simplify in v primeru, da se poenostavi v True ali False vrne že poenostavljen izraz, v nasprotnem primeru pa pokliče pomožno funkcijo toTseytin, ki sprejme seznam spremenljivk in izraz ter vrne trojico, ki je sestavljena iz spremenljivke, seznama preostalih prostih spremenljivk ter seznama s transformiranimi izrazi. Funkcija seznam spremenljivk pred klicem toTseytin preveri, da ne vsebuje neželenih ali podvojenih prostih spremenljivk. Rezultat funkcije toTseytin se predela v KNO s pomočjo funkcije map in kno, ki sprejme izraz v obliki tseytin transformacije in vrne izraz v KNO. Na dobljenem seznamu se nato uporabi še pomožno funkcijo glue, ki seznam seznamo pretvori v seznam. Na koncu se seznam, ki ga vrne glue obda z izrazom And ter s tem dobljen poenostavljen izraz vrne.

Pomožna funkcija toTseytin je definirana z ujemanjem vzorcev, uporablja pa tudi pomožno funkcijo tseytinAndOr, ki obdela seznam podobno kot to naredi funkcija toTseytin. Funkcija checkDuplicates s pomočjo funkcije exists preveri, da se spremenljivka ne pojavi v preostalem seznamu ali izrazu. Funkcija kno pa s pomočjo ujemanja vzorcev pretvori izraze, ki jih vrne tseytin transformacija v KNO. Uporabi se tudi funkcija exprToVarList, ki sprejme izraz in vrne seznam s spremenljivkami, ki se v izrazu pojavijo.

SATsolver:

Metoda sprejme seznam spremenljivk in izraz ter vrne NONE v primeru, da ta nima rešitve, čene pa SOME l, kjer je l seznam parov tipa 'a*bool. Funkcija najprej preveri, če je izraz v KNO in če v njej ni, ga v njo transformira s pomočjo metode tseytinTransformation in seznama novih spremenljivk. Preverba je pri tem napisana po nivojih, saj v nasprotnem primeru težko zagotovimo, da je izraz resnično v KNO. Izraz v KNO prejme pomožna funkcija stepOne, ki najde proste spremenljivke s pomožno funkcijo findFree in jih preveri, da niso v protislovju s pomočjo funkcije checkFree. V primeru protislovja vrnemo NONE, v primeru praznega seznama se kliče pomožna funkcija stepTwo, čene pa stepOne ponovimo. Funkcija stepTwo s pomočjo ujemanja vzorcev preveri, če smo našli rešitev oziroma da rešitve ni in v tem primeru algoritem konča, čene pa kliče stepThree. Ta najde prosto spremenljivko s pomočjo funkcije findVar in ji poiskuša pripisati vrednost true oziroma false, nato pa pokliče stepOne. Funkciji stepOne in stepThree kličeta pomožno funkcijo simplifyExpr, ki sprejme seznam spremenljivk in njihovi vrednosti ter izraz, vrne pa poenostavljen izraz, v katerem so spremenljivke zamenjane.

equivalentExpressions:

Metoda sprejme dva izraza ter vrne true, če sta izraza ekvivalentna in false čene. Funkcijo sem implementiral tako, da ta iz izrazov sestavi nov izraz ter v njem preimenuje spremenljivke s pomočjo pomožne funkcije aExpToIntExp. Nato pomožna funkcija count prešteje število spremenljivk, ki jih bo izraz potreboval za tseytin transformacijo in te spremenljivke ustvari. Na izrazu se nato naredi tseytinTransformation ter dobljen izraz v KNO pošlje v SATsolver, ki vrne opcijo s tipom seznam. Če je opcija enaka NONE, sta izraza ekvivalentna, čene nista. Funkcija aExpToIntExp sprejme izraz polimorfnega tipa in vrne izraz tipa int. To naredi tako, da najde spremenljivke v izrazu, jim pripiše vrednost ter jih nato v izrazu spremeni v pripisano vrednost. Funkcija equivalentExpressions uporablja funkcije aExpToIntExp, tseytinTransformation, SATsolver, foldl in isSome, v lokalnem okolju pa ima definirani funkciji count in createVarList.

problemReduction:

Metoda sprejme timetable in seznam tipa student in vrne izraz v KNO, s katerim lahko najdemo urnike za študente, ki ustrezajo določenim pogojem. To sem naredil s splošno funkcijo problemReduction N, ki sprejme še dolžino cikla vaj. ProblemReduction je tako implementiran kot problemReductionN 1. Pogoje sem razdelil na pomožne funkcije, ki naredijo posamezen pogoj. Prvi pogoj se pri tem izvede v lokalnem okolju, saj si pri drugem in trejem pogoju pomagamo s spremenljivkami, ki jih ta pogoj vsebuje. Drugi in tretji pogoj sta si podobna, zato sem jih implementiral s splošno funkcijo, ki sprejme funkcijo za filtriranje spremenljivk iz seznama. Metoda uporablja dodatne funkcije rmDup, exprToVarList, map in filter. V lokalnem okolju pa ima poleg že omenjenih funkcij še funkcijo, ki gre čez študente in na vsakem posameznem izvede prvi pogoj.

solutionRepresentation:

Metoda sprejme opcijo tipa seznam in vrne seznam z pari, ki vsebujejo tip student in timetable. Če je argument, ki ga sprejme NONE, fukcija vrne prazen seznam, čene pa prefiltrira seznam z rešitvijo in pošlje v pomožno funkcijo oneStudent seznam s spremenljivkami, ki so imele v rešitvi vrednost true. Pomožna funkcija seznam razdeli na seznam spremenljivk enega študenta in seznam preostalih spremenljivk. Seznam s spremenljivkami enega študenta pretvori v par student*timetable in ga doda v seznam, ki ga vrne rekurzivni klic na preostanku seznama. Metoda uporablja pomožne funkcije rmDup, map in filter.

Testiranje

Struktura funkcij eval, removeEmpty, removeConstants, pushNegations in removeVars je taka, da lahko pretestiramo vse vrste podizrazov in s tem pretestiramo večino možnosti za napako. Pri testiranju teh metod sem zato napisal večino možnih podizrazov, ki se lahko pojavijo znotraj izraza. Ob takem testiranju in poznavanju delovanja metod lahko predpostavimo, da s tem pretestiramo vse potrebno, a sem poleg tega vseeno dodal še nekaj "daljših" primerov. Metoda simplify uporablja metode removeEmpty, removeConstants, pushNegations in removeVars, zato lahko že pred testiranjem predpostavimo, da bo pravilno poenostavila vse primere, ki jih pretestiramo v prej omenjenih metodah. Poleg tega sem dodatno potestiral nekaj primerov, ki jih druge metode ne pretestirajo in ki so malo "daljši".

Pri testiranju funkcije tseytinTransformation je najpomembneje, da pretestiramo robne primere, izjeme ter kakšen daljši primer. Pri testiranju te funkcije sem zato preiskusil vse izjeme, ki jih funkcija sproži, transformacijo vsakega tipa izraza ter daljši primer. Funkcija tseytin se uporabi tudi v funkciji SATsolver, zato v tej funkciji ni potrebno preizkušati izjeme, ki so povezane s tseytin transformacijo. Tako kot tseytinTransformation, sem tudi v funkciji SATsolver preiskusil vse tipe izrazev, robne primere in daljši primer. Pri testiranju funkcije equivalentExpressions sem presikusil robne primere ter nekaj lažjih ter težjih izrazov.

Formula, ki jo vrne funkcija problemReduction je zelo nepregledna, zato sem jo stestiral skupaj z metodo solutionRepresentation. Tukaj preizkusil le nekaj robnih primerov ter nekaj malo daljših testov.