равномерным растяжением расстояний кислых протонов от соответствующих гетероатомов (рисунок 5).

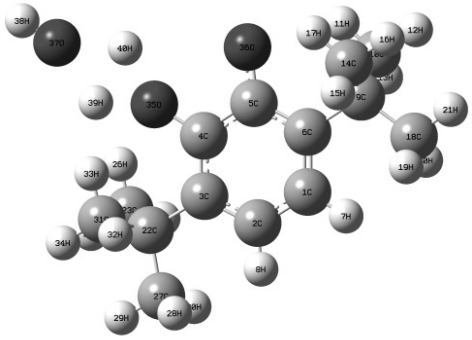

Рисунок 5. Структура, отвечающая максимуму полной энергии на ППЭ
3,6-дитрет.бутил-2-оксифеноксил – вода

В частности указанные величины для данной точки получились равными:

$$R(OH)_{3,6\text{-дитрет.бутил-2-оксифеноксил}} \approx 1{,}49\ Å; \qquad R(OH)_{H2O} \approx 1{,}62\ Å .$$

Таким образом, топологический анализ трехмерной ППЭ модельной межмолекулярной реакции протонного обмена между 3,6-дитрет.бутил-2-оксифеноксилом и водой позволяет выявить 4 особых точки на потенциальной поверхности. Точки А и В соответствуют реагентам и продуктам реакции. Точки С и C_1 соответствуют предельным ионизированным структурам, т.е. циклическим ИКВС, образующимся при соответствующем внутрикомплексном протонировании компонентов реакции.

Литература

1. Масалимов А.С., Ергалиева Э.М., Тур А.А., Рахимов Р.Р., Прокофьев А.И. Квантово-химическое исследование механизма реакций протонированияразличных молекул 3,6-ди-трет.бутил-2-оксифеноксилом. // Вестник Карагандинского университета. - Серия Химия.- 2013. - №3. - С.34-42.

2. Kurmanova A.F., KutzhanovaK.Zh., Pushchina A.V., Pustolaykina I.A Proton exchange in ammonia, water and formic acid dimers: quantum-chemical calculation// Вестник Карагандинского университета. - Серия химия. – 2018. - №2 (90). – С.64-71.

3. Ыбрахым С.А., Курманова А.Ф., Кутжанова К.Ж., Пустолайкина И.А. «Построение поверхности потенциальной энергии межмолекулярного протонного обмена в системе вода- 2-оксифеноксил»: Инновационное развитие и востребованность науки в Современном Казахстане: Материалы XIII Межд.науч.конф. -Тараз, 2019.- С.135-138.

Мухамедиева Л.С., Л.Н. Гумилев атындағы ЕҰУ, ақпараттық технология факультеті, М1-10020-03, магистрант
(Ғылыми жетекші – ф.-м.ғ.к Туребаева Р.Д.)

ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕР ДРАЙВЕРЛЕРІН ВЕРИФИКАЦИЯЛАУ

Құрылғы драйверлері - бұл жоғары деңгейлі бағдарламалық жасақтама немесе компьютерлік бағдарламалар аппараттық құралмен байланысуға мүмкіндік беретін бағдарламалар. Бағдарламалық жасақтаманың бұл компоненттері құрылғылар мен амалдық жүйелер арасындағы байланыс ретінде жұмыс істейді, осы жүйелердің әрқайсысымен байланысады және командаларды орындайды. Олар жоғарыдағы бағдарламалық жасақтама үшін абстракция деңгейін қамтамасыз етеді, сонымен қатар операциялық жүйенің ядросы мен төмендегі құрылғылар арасындағы байланысты қамтамасыз етеді.

Драйверлер операциялық жүйе ядросының ең көп бөлігін (70% дейін) алады. Бүкіл ОЖ-нің дұрыс жұмыс істемеуіне әкеліп соғатын, драйверлердің бастапқы кодында көптеген түрлі қателер болуы мүмкін. Linux ОЖ-сін 2000-шы жылдардың басындағы зерттеулеріндегі ядролық нұсқаларында 1.0-тен 2.4.1-ге дейін жүргізілген зерттеулер нәтижелері көрсеткендей, ядродағы барлық қателердің 85% -ын драйверлер құрайды екен. Ал 2006 жылғы Microsoft Windows XP

ОЖ ядросы үшін де осындай зерттеулерде ОЖ ядросындағы қателіктердің көбісі драйвер құрылғысында болатыны көрсетті. 2011 жылы Linux ядросының 2.6.0-ден 2.6.33-ке дейінгі нұсқаларында жүргізілген зерттеулер көрсеткендей, әртүрлі архитектуралар мен файлдық жүйелерді қолдауға жауапты ядро компоненттеріндегі қателіктерге қарағанда драйверлердегі қателіктердің саны азайғаны мен де әлі көп.

Драйвердегі қателіктерді шартты түрде типтік және типтік емес деп бөлуге болады. Типтік емес қателерге тиісті құрылғылармен және драйвердің интерфейс келісім-шарттарын бұзумен байланысты қателер жатады. Типтік емес қателер басқа қателер арасында ерекшеленеді, олар тек кейбір драйверлер үшін тән константтар, шектеулер, есептеулер және т.б. Типтік қателіктердің ерекшелігі олар үшін барлық драйверлер үшін немесе драйверлер тобы үшін ортақ тиісті ережелерді бөлуге болады.

Типтік қателер арасында үш классты бөлуге болады. Бірінші классқа Си бағдарламалау тіліндегі бағдарламаларда сияқты драйверлердегі жалпы қателер кіреді (Linux ОЖ ядросы мен драйверлері Си бағдарламалау тілінде әзірленеді). Мысалы, нөлдік көрсеткішті теріске шығару, бүтін түрдегі және т. б. айнымалылардың максималды ықтимал мәндерін арттыру. Екінші класқа ядро интерфейсін дұрыс пайдаланбаумен байланысты арнайы қателер жатады. Мұндай қателердің қатарына, мысалы, ерекше типтері бар айнымалыларды инициализациялау ережелерін бұзу жатады. Үшінші класс-синхрондау механизмдерін пайдаланбау немесе дұрыс пайдаланбау салдарынан параллель орындау кезінде пайда болатын жарыс және өзара бұғаттау күйлері.

Типтік жалпы қателер үшін жіктеу (классификация) бар. Олардың бұзылуы салдарынан болатын ережелер уақыт өте келе іс жүзінде өзгермейді (жалпы айтқанда, Си бағдарламалау тілі және оның кеңеюі дамуын жалғастырады, бұл Жалпы ережелерде шамалы өзгерістерге әкеледі). Параллельді орындаумен байланысты қателер, сондай-ақ стандартты және аз уақыт аралығында өзгереді.

Типтік ерекше қателер стандартты емес. Оларға сәйкес келетін ережелер өзгереді, жойылады және жиі пайда болады, себебі Linux ОЖ ядросының дамуы үлкен қарқынмен жүргізіледі, ядроның өзегі интерфейсі де өзгереді. Linux ОЖ ядросының жетекші әзірлеушілерінің бірі Грег Кроа-Хартманның мәлімдемесі бойынша, ядроның негізгі бөлігінің интерфейсі тұрақсыз болып табылады.

Анализ жасау үшін қажетті деректер осы құралдарда драйвер кодын орындау кезінде жиналады. Сондықтан құралдар драйверге арналған жабдықтың болуын немесе осы жабдықтың эмуляциясын талап етеді. Екінші жағынан, талданатын драйверлердің бастапқы коды әдетте талап етілмейді. Бұл жабық бастапқы коды бар драйверлерді талдауға мүмкіндік береді. Динамикалық верификация құралдары әдетте статикалық талдау жүйесіне қарағанда аз жалған іске қосылуларды береді, бірақ қазіргі уақытта орындаудың бір ғана жолын тексереді. Статикалық талдау құралдары, керісінше, драйвердің кодындағы көптеген (немесе тіпті барлығы) орындау жолдарын бір уақытта тексереді, бірақ динамикалық талдауды қолданғаннан гөрі олардың жалған іске қосылуы мүмкін.

Статикалық талдау дегеніміз бағдарламаны оның қауіпсіздіктің сыни қасиеттеріне сәйкестігін тексеру үшін талдауды білдіреді. Мысалы, жүйелік бағдарламалық жасақтама «ядролық деректер құрылымына жазбас бұрын пайдаланушының рұқсаттарын тексеру», «тексерусіз бос көрсеткішке сілтеме жасамау», «буфердің толып кетуіне тыйым салу» және т.б. ережелерге сәйкес болуы керек. Мұндай тексерулер кодтың нақты орындалуынсыз тексерілуі мүмкін.

Статикалық талдаудың жалпы мақсатты құралдары компиляторларды әзірлеу саласында пайда болған. Олар компилятордың жұмыс жылдамдығымен салыстырылатын жұмыс жылдамдығына ие. Бұл құралдар ағаштардағы қате құрылымдарды іздеуге негізделген синтаксистік шығыс әдістерімен абстрактілі интерпретациясы бар деректер ағындарын талдау әдістері қолданады. Соңғылары мәндердің кейбір көп қажетсінетін жиыны бағдарлама конструкциялары бойынша, бұл көп "толмайынша" техпорға дейін тарайтынына негізделген, яғни келесі тарату кезінде өзгермейді.

Типтік қателерді анықтау үшінжалпы мақсатты статикалық талдау құралдары өте пайдалы, олар арнайы бағдарламалау тіліне ортақ.

Кодтың статикалық талдауының алғашқы жалпы мақсатты құралдары 70-ші жылдары пайда болды, мысалы, Bell Labs компаниясында Си бағдарламаларды талдау үшін Lint құралы әзірленді. Құрал Си тіліне тән қате құрылымдарды табуға қабілетті болды, соның ішінде,

инициализацияланбаған айнымалыларды пайдалану, жағдайлардың ішіндегі константаларды қате пайдалану, нәтижесі қорытындыға сәйкес келмейтін есептеулер ситяқты. Осы тексерулердің көпшілігі тікелей қазіргі заманғы компиляторлармен жүзеге асырылады.

Бүгінгі күні бағдарламаларды индустриялық әзірлеуде кеңінен қолданылатын статистикалық талдаудың әртүрлі жалпы мақсатты құралдарының көп саны бар. Мұнда Linux ОЖ ядросына қолдану тәжірибесі бар құралдар қарастырылған.

Sparse құралы Linux ядросының ішінде дамыды. Оны 2003 жылы Linux негізін қалаушы Торвальдс Линусын дамыта бастады. Одан әрі оны әзірлеуді Josh Triplett және Christopher Li ядросының әзірлеушілері жалғастырды. Құрал синтаксистік әдістерге негізделген.

Sparse келесі қателерді табады:

1. Типін шектеулі типке түрлендіру немесе шектеулі типті жарамсыз өрнектерде пайдалану (bitwise, casttruncate, default-bitfield-sign, enum-mismatch, one-bit-signed-bitfield, ptr-subtraction-blows, typesign).
2. Декларацияларды пайдалану: статистикалық айнымалылар (decl), c89 стандарты бойыншаблоктың басынан тыс айнымалы декларация (declaration-afterstatement), декларацияларды жасыру (shadow), transparent-union атрибуттарын пайдалану (transparent-union).
3. Адрестік кеңістіктегі көрсеткіштерді пайдалану (address-space, coast-to-as).
4. Ескі синтаксисті құрылымдарды инициализациялау (oldinitializer) немесе массивтерді инициализациялау (parenstring) кезінде GCC әлсіреуін пайдалану. Аннотирование контекста использования функций, в частности, использованиесинхронизационных примитивов (no-context).
5. Цикл блогы үшін жақшаның болмауы (do-while).
6. Нөлдік көрсеткіш ретінде нөлді пайдалану (non pointer-null).
7. Функциялардың қайтарылатын мәндерін тексеру (return-void).
8. Макрос параметрлерін пайдалану (undef).

Тексерулерде құрал **GCC** кеңейтімдерін пайдалана отырып, ядроның бастапқы кодындағы аннотацияға сүйенеді. Келтірілген ережелерде кем дегенде бес түрлі кеңейтулер көрсетілген. Сондықтан осы кеңейтулерді қолдау Linux ОЖ ядросына бағытталған статикалық талдау құралы үшін аса маңызды болып табылады, өйткені бұл атрибуттар статикалық талдауышқа қосымша кеңестер береді. Ядролардың 2.6.25, 2.6.29, 2.6.30 нұсқаларында қателерді табу үшін Sparse құралын қолданған. Онда кем дегенде 140 қате табылды. [1]

Stanse құралы блоктау функцияларын дұрыс пайдаланбауға, жадыны дұрыс пайдаланбауға, атомдық контекст ішіндегі ықтимал ұйықтайтын (sleep) функцияларға, қол жетпейтін кодқа, үзумен жұмыс істеу функцияларын дұрыс пайдаланбауға (irq disable/enable), pci, tty құрылғылар құрылымының сілтемелерін есептеуге байланысты қателерді таба алады.Құрал деректер ағындарынпроцедурааралық талдауға негізделген. 2009-2010 жылдары құрал Linux ядросына қолданылды, шамамен 130 қате табылды.

Coverity құралы [2] келесідей қате түрлерін табады, олар: қол жетпейтін код, нөлдік көрсеткіштерді ойнату, салыстыруға дейін пайдалану, буфердің толып кетуі, ресурстардың ағып кетуі, қайтарылатын мәндердің қауіпсіз пайдаланбауы, тип өлшемдерінің сәйкес келмеуі және бөлінетін жадтың бөлінбеуі, куәландырылмаған айнымалыларды оқу және босатылғаннан кейін жадты пайдалану. Бұл қателерді әртүрлі интерфейстерге бейімдеуге болады, мысалы, Linux ядросында ресурстарды бөлу функциясы ретінде USB ішкі жүйесінің urb құрылымына жадты бөлу функциясы қарастырылады.

Құрал Linux ОЖ ядросына 2006 және 2009 жылдар аралығында, АҚШ қауіпсіздік департаментімен келісім-шарт аясында қолданылған [3].

Жұмыс авторлары барлық ескертулерден қателіктер туралы 2,125 хабарлама таңдалғаны туралы мәлімет береді. Барлық осы хабарлар ядро әзірлеушілеріне жіберілді. Алайда олардың 61% ғана өңдеуге қабылданды. Жалған ескертулер мен түзетілген қателер саны туралы мәліметтер келтірілмейді.

Coccinelle құралы бастапқыда кодтағы өзгерістерді қолдауды мақсат етіп қойды, осылайша, өзгерту үлгісі сипатталды, әрі қарай бұл үлгі барлық бастапқы кодқа қолданылды. Linux ОЖ ядросында, мысалы, функцияның атауын өзгерту, контекст бойынша шектеулері бар функцияның аргументін қосу, деректер құрылымын қайта құру сияқты өзгерістер болып табылады. Сондай-ақ, құрал өзгерістерді қолдау сияқты үлгілермен сипатталатын ядрода қателерді іздеу үшін қолданылады, бірақ өзгерістерді қолданудың орнына қате туралы хабар

шығады. Linux OS ядросына Coccinelle қолдану табыстылығы қате іздеген Үлгіге байланысты. Кейбір шаблондар үшін жұмыста жалған іске қосылулар саны 98% (буфердің толып кету қателері) құрады, бұл көп болмайды. Қате үлгісі үшін босату кейін пайдалану жалған ескертулер саны 60% құрады, бұл әлдеқайда жақсы, бірақ әлі де көп. Мақаласында [4] авторлар ең аз жалған іске қосу үлгілерін таңдады. Нәтижесінде жалған іске қосылудың орташа саны 15% - дан аз болды.Coccinelle көмегімен барлығы 360 қате табылды, оның 30 Linux ядросында түзетілді.

Saturn құралы [5, 6] функциялардың аннотацияларын шығаруға негізделген, ведомствоаралық талдауды жүзеге асырады. Әрбір нақты функцияны талдау үшін жол формуласын құру және SAT торларды пайдалану сияқты жоғары дәлдікті әдістер қолданылады. Процедурааралық талдау функциялардың аннотациясын ғана пайдаланады.

Авторлар функциялар арасындағы тәуелділікті қоспағанда, әрбір функцияны жеке талдау есебінен табиғи түрде алынатын талдаудың параллельденуіне көп көңіл бөледі. 40-100 процессорлардың ішінен кластерде іске қосу тиімділіктің 80-90% - ын параллель береді, нәтижесінде Linux ОЖ ядросын талдау бірнеше сағат алады,бұл SAT-ды пайдаланбайтын басқа жалпы мақсатты амалдармен салыстыруға болады.

Мақалада [7] жоғары дәлдікті әдістерді қолдану процедурааралық тәуелділікті талдауға дейін кеңейтілген. **Saturn** құралын 2.6.17.1 версиясындағы Linux ОЖ ядросының бастапқы кодына қолдану туралы мәліметтер келтіріледі. Атап айтқанда, жұмыста дәлірек талдауды қолдану қате саны өзгермеген жағдайда (134) жалған іске қосылулар санын 1344-тен 37-ге дейін айтарлықтай төмендетуге мүмкіндік беретіні көрсетілген.

Locksmith құралы [8] параллель орындау кезінде жарыс жағдайларына байланысты қателерді іздейтін жалпы мақсатты талдаудың бірнеше құралдарының бірі. Жұмыста Linux ядросының он драйверіне құралды қолдану туралы деректер келтіріледі, алайда табылған қателер туралы ештеңе хабарланбайды.

Сондай-ақ, жалпы мақсатты статикалық талдаудың көптеген құралдары бар, мысалы, **Klocwork Insight** [9] коммерциялық құралы, **FindBugs**, **Splint**, **Svace** және т.б. академикикалық құралдары.

Барлық жалпы мақсатты статикалық талдағыштар көптеген Си бағдарламаларға тән жалпы қателерді анықтауға бағытталған.

Барлық жалпы мақсатты статикалық талдағыштар ӨҚ көптеген бағдарламаларға тән облыстық қателерді анықтауға бағытталған. Бірақ олар драйверлерді талдау ерекшелігін ескермейді, өйткені драйверлер мен ОЖ ядросының өзара іс-қимыл логикасы туралы білімі жоқ. Осы себепті олар осындай логиканың немесе драйверлердің ОЖ ядросымен өзара іс-қимыл жасау ережелерінің бұзылу қателерін анықтай алмайды.

Барлық осындай жүйелер драйверлердің ОЖ ядросымен өзара әрекеттесу ережелерін бұзумен байланысты қателерді іздеуге арналған. Осыған байланысты, драйвердің арнайы ортасын дайындау қажет, бұл драйверді өңдеушінің шақыру тәртібіне қойылатын шектеулерді ескеруге мүмкіндік береді. Верификациялау үшін бағдарламаның еркін нүктесінің жетістіктерін тексеруге мүмкіндік беретін статикалық верификация құралдары қолданылады.

Операциялық жүйелердің драйверлерін верификациялаудың ең дамыған жүйесі Microsoft SDV жүйесі болып табылады, ол Microsoft Windows операциялық жүйесінің драйверлерін статикалық верификациялауды жүргізуге асыруға мүмкіндік береді. Ол драйверлерді сертификаттау процесінде қолданылады және 2006 жылдан бастап Microsoft Windows Driver Developer Kit құрамына енгізілді. Microsoft SDV жүйесі нақты бағдарламаларды верификациялауды қолдану мүмкіндігін көрнекі көрсетеді.

SDV жүйесінде пайдаланушыдан қоршаған ортаны құру үшін, өңдеушілердің әрбір функциясының рөлін көрсете отырып, драйверлердің бастапқы кодын қолмен аннотациялауды талап етеді.

Аннотацияларды қолмен жасау керек, алайда, SDV қолданудан оң нәтиже аннотациялауға еңбек шығындарын өтейді, сондықтан көбінесе аннотацияларды драйверлерді әзірлеушілер тікелей жазады. SDV тәсілі шеңберінде қоршаған ортаның моделі ретінде өңдеушілер функциясының түрлеріне сәйкес шақырулардың еркін реттілігі жасалады. Генерацияланатын тізбекте өңдеушілердің түрлеріне сәйкес функцияларды шақыру тәртібіне қойылатын шектеулер ескеріледі. Microsoft Windows ОЖ ядросының негізгі бөлігінің функциялары қарапайым аналогтармен модельделеді, және де олар үшін әдетте детерминацияланбаған модельдер қолданылады. Сонымен қатар, статикалық талдау құралы

недетерминизмнің барлық тармақтарын зерттейді, сонымен сирек кездесетін функцияларды қоса алғанда, функциялардың барлық әр түрлі нұсқаларын тексереді.

Тексерілетін әдептілік ережелері SLIC тіл көмегімен қалыптастырылады, онда драйвердің бастапқы кодымен байланыс ядро функциясының шақыруларын ұстап тұратын аспект-бағытталған конструкциялардың көмегімен орнатылады. Бұл тәсілде қате моделін әзірлеу үшін Си бағдарламалау тіліне өте ұқсас қарапайым арнайы тіл беріледі. Бұл тіл функциялардың мінез-құлқын моделдеуге ғана бағытталған, сондай-ақ бағдарламаның жаһандық жағдайын сақтауға мүмкіндігі бар. Жалпы айтқанда, осының салдарынан қателерді модельдеу үшін қажетті мәнерлі қуат өте қатты зардап шегеді.

```
state { int zero_cnt = 0; }
put.entry {
  if ($1 == 0) {
    if (zero_cnt == 4)
      abort "Queue has 4 zeroes!";
    else
      zero_cnt = zero_cnt + 1;
  }
}
get.exit {
  if ($return == 0)
    zero_cnt = zero_cnt - 1;
}
```

Сур.1 - SLIC үлгі ерекшелігі

Алайда, тілдің қарапайымдылығы оны верификация саласындағы сарапшылар емес пайдаланушыларға да қате модельдерін құру үшін пайдалануға мүмкіндік береді. Қазіргі уақытта шамамен 200 ережелердің жинағы таңдалған, ал зерттеу нұсқасында жаңа ережелерді қосу мүмкіндігі қосылды.

Келтірілген 1-суретте көрсетілген SLIC тіліндегі ерекшелігі жасанды қатені тексеруді көрсетеді.

Бұл жағдайда, кезекте төрт нөлден артық жағдай қате болып табылады.SLIC препроцессоры спецификация негізінде драйверлердің бастапқы кодын құралдайды. Нәтижесінде, құрал Си-ге баламалы бағдарламаны генерациялайды, содан кейін кодты статикалық талдау құралы арқылы тексеріледі.

Өкінішке орай, SLIC тек Microsoft Windows ОЖ драйверлері үшін спецификация мен құрал-саймандарды жасау үшін пайдалануға болады. Сонымен қатар, SDV жүйесін іске асырудың бастапқы коды толығымен жабық. Жүйеге SLAM және Yogi статикалық верификация құралдары біріктірілген.

Сонымен қатар, Windows-тің драйвері мен ОЖ арасындағы интерфейс өте сирек өзгереді, жүйенің архитектурасын әзірлеу кезінде драйверлер мен ОЖ ядросы арасындағы интерфейстің тұрақты даму мүмкіндігін ескеруді талап етпейді.

Пайдаланылған әдебиеттер тізімі:

1. Palix N., Thomas G., Saha S. et al. Faults in linux: ten years later // SIGPLAN Not. 2011. Vol. 47, no. 4. P. 305–318.

2. Engler D., Chelf B., Chou A. Checking system rules using system-specifc, programmer-written compiler extensions // Proceedings of the 4th conference on Symposium on Operating System Design & Implementation - Volume 4.2000. P. 1–16.

3. Guo P. J., Engler D. Linux kernel developer responses to static analysis bugreports // Proceedings of the 2009 conference on USENIX Annual technical conference. USENIX'09. Berkeley, CA, USA: USENIX Association, 2009.P. 22–22.D.

4. Lawall J. L., Brunel J., Palix N. et al. WYSIWIB: A declarative approach to fnding API protocols and bugs in Linux code // DSN'09 – The 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. 2009. P. 43–52.

5. Xie Y., Aiken A. Saturn: a SAT-based tool for bug detection // Proceedings of the 17th international conference on Computer Aided Verifcation. CAV'05. Berlin, Heidelberg: Springer-Verlag, 2005. P. 139–143.
6. Xie Y., Aiken A. Saturn: A scalable framework for error detection using Boolean satisfability // ACM Trans. Program. Lang. Syst. 2007. Vol. 29, no. 3
7. Dillig I., Dillig T., Aiken A. Sound, complete and scalable path-sensitive analysis // SIGPLAN Not. 2008. — June. Vol. 43. P. 270–280.
8. Pratikakis P., Foster J. S., Hicks M. LOCKSMITH: Practical static race detection for C // ACM Trans. Program. Lang. Syst. 2011. Vol. 33, no. 1. P. 3:1–3:55.
9. Сыромятников С. Декларативный интерфейс поиска дефектов по синтаксическим деревьям: язык KAST // Труды Института системного программирования РАН. 2011. Т. 20. С. 51–68.

Мұхамедия А.М., академик Е.А. Бөкетов атындағы Қарағанды мемлекеттік университеті, Математика және ақпараттық технологиялар факультеті, ММатО-51 тобы, магистрант
(Ғылыми жетекшісі — п.ғ.к., доцент Шаяхметова Б.К.)

МАТЕМАТИКАНЫ ОҚЫТУДА ПРАКТИКАЛЫҚ МАЗМҰНЫ БАР ТАПСЫРМАЛАРДЫ ҚОЛДАНУ

Аннотация: Оқушылардың жалпы математикалық функционалдық сауаттылығын қалыптастыруды математика сабақтарына күнделікті өмірмен байланысты практикалық сипаттағы есептерді кіріктіру арқылы қол жеткізуге болатындығы айтылады.

Тірек сөздер: Қолданбалы сипаттағы математикалық есептер; экономикалық мазмұнды есептер; өмірде, тұрмыста кездесетін есептер; өндіріс; өнімділік, тиімділік.

Аннотация: В статье говорятся о том, что формирование общей математической функциональной грамотности учащихся может быть достигнуто путем интеграции практических уроков, связанных с повседневной жизнью, в уроки математики.

Ключевые слова: Математические задачи прикладного характера; экономически значимые задачи; задачив встречаемые в жизни, в производстве; производительность; эффективность.

Annotation: It is said that the formation of general mathematical functional literacy among students can be achieved by integrating practical lessons.

Key words: Mathematical problems of applied nature; economically significant tasks; problems in like; production; performance; efficiency.

Заманауи математикалық білім беруді модернизациялаудағы негізгі жағдайдың бірі – жалпы білім берудің барлық деңгейлерінде математика курсының практикалық бағытын күшейту, яғни оның мазмұны мен оқыту әдістерінің практикамен байланысын жүзеге асыру. Бұл жаңа мәселе емес. Өзінің қалыптасуы мен дамуы барысында көптеген сұрақтар қойылған, олардың кейбіреулері әлі күнге дейін шешімін таппаған. Мектеп математикасын практикалық бағыттау мәселесі математикалық теорияның үнемі дамып отыруына, компьютерлік технологияның дамуына, адам қызметінің өрісінің кеңеюіне байланысты қарқынды өзгерістерге түсіп отырады. Тіпті мәселе бір рет шешілген болса да, тарихтың әр жаңа кезеңімен қайта ойластырып, түзетуді қажет етеді.

Оқушылардың болашақ іс-әрекетінде математиканы қолданудың барлық аспектілерін болжау мүмкін емес және бұл мәселелерді білім беру ұйымында қарастыру одан да қиын. Адам қызметінің барлық бағыттарындағы ғылыми-техникалық жетістіктер білімге, техникалық мәдениетке, білім берудің жалпы және қолданбалы сипатына жаңа талаптар қояды. Бұл қазіргі заманғы мектеп үшін оқушыларды практикалық қызметке дайындауда жаңа міндет қояды [1, 272 б.].

Практикалық мазмұны бар математикалық есеп (қолданбалы есеп) – бұл басқа пәндермен сабақтаса жүретін, заманауи өндірістің технологиясы мен экономикасында, қызмет көрсету секторында, күнделікті өмірде, еңбек операцияларын орындау кезінде қолданылатын тапсырмалар. Қазіргі қолданыстағы математиканың негізгі бөлімдерінде математикалық модельдеу, алгоритмдеу және бағдарламалауға арналған есептер шығарылады.

Айналадағы шындықтың мысалдары оқушыларға математиканың практикалық маңыздылығын, оның тұжырымдарының кең тұтастығын ашуға мүмкіндік береді. Тәжірибе көрсеткендей, оқушылар практикалық мазмұндағы тапсырмаларды қызығушылықпен орындайды, теориялық мәселенің практикалық тапсырмадан қалай туындайтынын және таза