Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Szűcs Dániel

2023

Szegedi Tudományegyetem

Informatikai Intézet

Es2panda fordítóprogram fejlesztések

Szakdolgozat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Szűcs Dániel |  | Dr. Kiss Ákos |  |
|  | informatika szakos hallgató |  | egyetemi docens |  |

Szeged

2023

Feladatkiírás

A témavezető által megfogalmazott feladatkiírás. Önálló oldalon szerepel.

Tartalmi összefoglaló

* A téma megnevezése:

Es2panda fordítóprogram fejlesztése, az ETS nyelvi konformancia növelés érdekében.

* A megadott feladat megfogalmazása:

A cél, hogy az Es2panda fordítóprogram funkcionalitását kibővítsem a különböző komponensei fejlesztése által. A Checker komponensnek képesnek kell lennie különböző, az objektum orientált programozási paradigma által leírt szemantikai elemzésekre, továbbá az ETS nyelv specifikus szkóp feloldási szabályainak kezelésére. A Compiler komponensnek képesnek kell lennie a feltételes vezérlési szerkezet és a különböző bináris operátorok bytecode-ra fordítására, valamint függvény szkópokban a paraméterek és lokális változók regiszter allokálására és spillelésére.

* A megoldási mód:

Először is, a Compiler komponens egyszerűbb részein, a vezérlési szerkezeteken és az operátorokon, majd a Checker komponens OOP mintáinak szemantikai elemzésén dolgoztam, így átfogó ismeretet szereztem a komponensek felépítéséről. Ezek után a két komponenst párhuzamosan fejlesztettem, az új nyelvi elemek implementációs sajátosságainak megfelelően.

* Alkalmazott eszközök, módszerek:

Az Es2panda fejlesztése Ubuntu 22.04 operációs rendszeren folyik, a C++ nyelv 17-es szabványában. A fejlesztéshez Visual Studio Code fejlesztői környezetet használok. A fordításhoz az LLVM toolchain 14-es verzióját használom, debuggoláshoz a cgdb eszközt választottam.

* Elért eredmények:

Implementáltam egy robosztus rendszert a regiszter spillelésre, valamint a regiszterek statikus típusainak követésére, továbbá implementáltam a szkóp feloldást, amely felhasználóbarát fordítási hibaüzeneteket ad a fordítóprogram felhasználójának.

* Kulcsszavak:

fordítóprogram, szemantikai analízis, regiszter spillelés

Tartalomjegyzék

[Feladatkiírás 2](#_Toc132961183)

[Tartalmi összefoglaló 3](#_Toc132961184)

[Tartalomjegyzék 4](#_Toc132961185)

[1. Bevezetés 5](#_Toc132961186)

[2. Mi az az Es2panda? 6](#_Toc132961187)

[2.1. Mi az a fordítóprogram frontend? 6](#_Toc132961188)

[2.2. Mi az az Es2panda? 6](#_Toc132961189)

[2.3. Hogyan fejlesztjük az Es2pandát? 6](#_Toc132961190)

[3. Az Es2panda struktúrája 7](#_Toc132961191)

[3.1. A fizikai struktúra 7](#_Toc132961192)

[3.2. A logikai struktúra 8](#_Toc132961193)

[3.2.1. A Lexical Analyser (Lexer) 8](#_Toc132961194)

[3.2.2. A Parser 9](#_Toc132961195)

[3.2.3. A Binder 9](#_Toc132961196)

[3.2.4. A Semantical Checker 10](#_Toc132961197)

1. Bevezetés

Szakdolgozatom témájául az Es2panda fordítóprogramon végzett fejlesztéseimet szeretném bemutatni. Mivel a fordítóprogramon végzett munkáim rendívül szerteágazóak, így sok különböző problémával találkoztam a fejlesztés során, melyekre a megoldás nem mindig volt triviális a már meglévő szoftver architektúra mellett. A fejlesztés alatt a következők voltak a céljaim:

* Robosztus könnyen érthető kód írása, amely követi a C++ nyelv bevált gyakorlatait. Ezek magukba foglalják az explicit memória kezelése minimalizálását, a const korrektséget, a makró használat minimalizálását, valamint a komponensek szoros összekapcsolódásának elkerülését.
* A futásidei teljesítmény növelése, akár a fordítási idő romlása árán. Ez leginkább a template metaprogramozási gyakorlatokkal értem el, melyek drasztikusan javítottak a kód hívó oldalának egyszerűsítésében.
* Az implementált függvények és típusok interfészeinek egyszerűsítése, hogy minél nehezebb legyen őket rosszul használni.
* A modern fordítóprogramok által alkalmazott technikák kutatása és az Es2pandába való integrálásuk. Ezek magukba foglalják a különböző optimalizációkat és statikus analíziseket.

A következő oldalakban, betekintést adok az Es2panda általam fejlesztett részeibe, azok tervezési folyamataiba, és példákat mutatok a teljesítmény orientált gondolkodásmódra, amely elengedhetetlen egy fordítóprogram esetében. Először is bemutatom az Es2pandát jelenlegi állapotában, majd egy áttekintést nyújtok az általános struktúrája felett. Bemutatom az Es2panda hat nagy komponensét és rajtuk végzett munkámat, mindezt a komponensek függőségének sorrendjében, kezdve a belépéstől a kész bytecode-ig. Ezek után egy az Es2panda által fordított forráskód demót mutatok be, amely forráskód az általam fejlesztett részeket teszteli, majd kitérek a generált bytecode-ra. Végezetül kitérek az érdekesebb implementációs részletekre.

1. Mi az az Es2panda?

Ebben a fejezetben általánosan beszélek az Es2pandáról mint szoftverről.

* 1. Mi az a fordítóprogram frontend?

A fordítóprogram frontend egy olyan szoftver, amely egy előre meghatározott nyelvet vagy nyelveket egy köztes reprezentációra vagy más néven bytecode-ra fordítja, melyet később egy fordítóprogram backend tényleges gépiódra fordít, vagy egy interpreter futásidőben értelmez.

* 1. Mi az az Es2panda?

Az Es2panda egy fejlesztés alatt álló fordítóprogram frontend melyet C++ nyelven fejlesztünk. A program több különböző szkriptnyelvet támogat és párhuzamos fordítással rendelkezik.

* 1. Hogyan fejlesztjük az Es2pandát?

Az Es2panda fejlesztése Linux platformon történik, a projekt fordításához szükséges csomagokat egy bootstrap szkript tudja telepíteni Ubuntu 20.04-es és 22.04-es kiadásokra. A build rendszer a CMake 3.10-es verziója. A build konfigurálható gcc és clang fordítóprogrammal is, a támogatott verzió gcc esetében 11.1, clang esetében pedig 14.0. Az Es2panda 64 bites Unix alapó rendszerekre lefordítható, mind x86-64 mind arm64 architektúrára. A projekt a C++17-es nyelv szabványt használja.

A verzió kezelést biztosító szoftver a git. A forráskód megtalálható a Gitee weboldalon. A különböző toolchain komponensek, mint a runtime, az Es2panda, és a runtime nyelvi pluginjai (Ecmascript, ETS) különböző repositorykban érhetők el.

1. Az Es2panda struktúrája

Az Es2panda mind logikailag, mind fizikailag szegmentálva van a struktúráját illetően. Ez a szegmentáció segít átlátni az absztrakciókat, kezelni a dependenciákat, elrejteni az implementációs részleteket és optimalizálni a fordítási időt.

* 1. A fizikai struktúra

Az Es2panda a standard C/C++ header módszerrel van strukturálva. Az absztrakt szintaxis fa (AST) minden node-ja a saját header fájljában van, viszont az hat nagy logikai komponens javarészt monolitikusan egy-egy header fájlban található. Néhány függvény, amelyek különben implementálhatók lennének egy külön fordítási egységben (*.cpp*), mégis headerben van implementálva. Ennek az oka a fejlesztés elején a fordítási idő gyorsítása volt, valamint a hívások jobb optimalizálása fordítási egység határokon (*inlining)*. Ezeket a függvényeket a fejlesztés jelen szakaszában próbálom megszüntetni, ha egy módosításom érinti őket, ugyanis a támogatott fordítóprogramok már képesek a linkelési idejű optimalizációra (LTO), így a legtöbb belső szimbólum deduplikálásra kerül.

A projekt tartalmaz több olyan fájlt is, amelyeket a build rendszer fordítás előtt generál. Ezek a fájlok általában olyan kódot tartalmaznak, amelyek repetitívek, vagy könnyen generálhatók egyszerű markup nyelvekből. A projekt a YAML markup nyelvet használja, amit Ruby szkriptek dolgoznak fel, majd az Embedded Ruby (ERB) rendszer alakítja át megfelelő forrás fájllá, általában headerré. Néhány generált fájlra a logikai egységek részletezésénél külön ki fogok térni.

Az Es2panda két ELF állományból áll. Az egyik az *es2panda-lib.so* programönyvtár, ami a tényleges fordítási logikát tartalmazza, a másik maga az *es2panda* nevű futtatható állomány, ami dinamikusan linekelt az előző programönyvtárhoz. Ez a szeparáció lehetővé teszi, hogy a fordítóprogram a futtatható állomány nélkül is integrálható lehessen más rendszerekbe. Azonban természetéből adódóan a szeparáció ront az LTO hatékonyságán, mivel a programkönyvtár szimbólumait nem lehet eltávolítani a dinamikus linkelés miatt.

* 1. A logikai struktúra

Ebben a fejezetben röviden bemutatom a logikai egységeket, a mélyebb működésükre és implementációs részleteikre a saját fejezeteikben fogok kitérni, kivéve az első egység esetében. Mivel ennek az egységnek a fejlesztésében nem foglaltam kulcsfontosságú szerepet, így nem szentelek neki külön fejezetet. Az Es2panda fordítóprogram hat nagy logikai elemre van strukturálva. Ezek név szerint a Lexical Analyser, a Parser, a Binder, a Semantical Checker, a Code Generator, és az Emitter. Van egy hetedik elem is, amin az előbb említett logikai elemek dolgoznak, ez az absztrakt szintaxis fa (AST).

* + 1. A Lexical Analyser (Lexer)

A Lexical Analyser (Lexer) felelős a beolvasott forráskód tokenizációjáért. Az Es2panda indulásakor a Lexer az első komponens amelyik inicializálódik. A Lexer szolgáltatja a Parser számára a tokeneket és a sor, oszlop információt hibaüzenetek esetén. Ennek a komponensnek egy része, a kulcsszavak beolvasása egy YAML markup fájlból generálódik a build során.

A Lexerből a *Lexer::NextToken* függvénnyel lehet az aktuális tokent kikérni, ez előre is mozgatja a lexer fejét a következő tokenre. Lehetőség van az aktuális pozíción lévő tokent követő karakter megvizsgálására, ezt a *Lexer::Lookahead* függvény valósítja meg. Esetenként előfordul, hogy a parsolás során a Lexert vissza kell tekerni. Erre lehetőséget adnak a *Lexer::Save* és a *Lexer::Rewind* függvények.

A generált *keywords.h* nevű fordítási egység valósítja meg a tényleges tokenizációt. A generált fájl függvényei a *Keywords::Scan\_\** alakúak, ahol a ‘*\*’* a következő betűt jelöli. Például, ha az első függvény a *Keywords::Scan\_i* volt, akkor a következő hívott függvény négy különböző lehet. Lehet a *Keywords::Scan\_if* ami ha f-et whitespace követi felvesz egy ’if’ *Lexer::Token-t* ami kulcsszó. Hasonlóan lehet a *Keywords::Scan\_im és Keywords::Scan\_in* mely függvények az ’implements’ és ’interface’ kulcsszavakat próbálják szkennelni. Ha ezek közül egyik sem teljesül, akkor a negyedik esetben az előállított *Lexer::Token* azonosító (identifier) lesz.

Az **1. Kód Részlet** mutatja hogyan lehet egy kulcsszót felvenni a leíró YAML fájlba.

A *name* mező a kulcsszó konkrét formáját adja meg, amit a Lexer keresni fog a tokenizáció során. A *token* mező a *Lexer::Token-hez* tartozó *Lexer::TokenType* generált enum egyik mezőjének neve lesz. A *keyword* és *keyword\_like* mezők megadják, hogy melyik nyelvben hogyan viselkedik ez a kulcsszó. Mivel az Es2panda fordítóprogram több nyelvet is támogat ezért a Lexer nagy része újra használható a nyelvek közös pontjai között. Tehát ahogy azt **1. Kód Részlet** mutatja, az *async* kulcsszó az AssemblyScript és ETS nyelvek esetén hard kulcsszó, azaz minden kontextusban kulcsszó, míg Ecmascript és Typescript nyelvek esetén soft kulcsszó, azaz a megfelelő pozícióban kulcsszó más pozícióban pedig azonosító.

1. Kód Részlet: ’async’ kulcsszó YAML rekord.

keywords:

- name: 'async'

token: KEYW\_ASYNC

keyword: [as, ets]

keyword\_like: [js, ts]

* + 1. A Parser

A Parser logikai egység a Lexer által előállított tokenekből építi fel az absztrakt szintaxis fát. Mivel az implementációja egy LR(1) -es parser így ezt alulról felfelé teszi, először a levél node-okat állítja elő majd azok szülő node-jait, így haladva egészen a gyökér node-ig. A Parser hívja a Binder megfelelő függvényeit, hogy a létrehozott AST node-ok megfelelő szkópokba legyenek bekötve.

* + 1. A Binder

A Binder komponens a Parser komponenssel együtt dolgozva építi fel a szkóp fát. A szkópok *binder::Variable* pointereket kötnek nevekhez egy *std::unordered\_map-ben*. A *binder::Variable* igazából egy szimbólum mivel nem csak változóra mutathat, hanem típus névre is és a megfelelő AST node deklarációjára mutat. Minden egyes szkópban a feloldás úgy történik egyszerűbb esetben, hogy a szkóp fát alulról felfelé bejárjuk, mindig csak a szülőkön iterálva, és megkeressük név alapján a megfelelő *binder::Variable-t*.

* + 1. A Semantical Checker

A Semantical Checker az Es2panda legkomplexebb logikai egysége. Feladata a kód szemantikai analízise. Ez Ecmascript esetében gyakorlatilag egy üres operáció, minden más támogatott nyelv esetén viszont több feladatot is el kell látnia. Az Es2panda által támogatott nyelvek az Ecmascript kivételével mind rendelkeznek valamilyen típus rendszerrel. A típusok által közölt információból és a nyelvek által támogatott különböző programozási paradigmák szabályrendszeréből a fordítóprogramnak warning és error üzeneteket kell előállítania, ha valamilyen szemantikai szabály sértett. A Semantical Checker gyakorlatilag majdnem minden AST node-hoz rendel egy *checker::Type* példányt, amelyben leírja az adott típust és különböző típus relációkat tud felállítani. Például osztályok között inheritencia láncot tud felállítani, vagy primitív típusok közötti konverziókat tud a primitívek mérete alapján engedélyezni vagy letiltani.

* + 1. A Code Generator

A Code Generator az absztrakt szintaxis fából, és a Semantical Checker által előállított típus információból generál bytecode-ot. Ez a bytecode statikusan típusos, kivéve Ecmascript esetén, ahol a runtime egy plugin segítségével támogatja a dinamikus bytecode-ot. A Code Generator párhuzamosan fut a fordítási egységekben lévő függvényekre. Ezt a működést mutexek nélkül tudja végezni (kivéve a szál scheduling esetében). A bytecode gyakorlatilag egy regiszter alapú veremgép. Egy regiszter az akkumulátor szerepét tölti be. A Code Generator felelős a regiszterek megfelelő kiválasztásáért a regiszter spillelésért. A Code Generator egy vectorban tárolja el a kigenerált utasításokat.

* + 1. Az Emitter

Az Emitter komponens a kigenerált utasítás vectort használja fel. Gyakorlatilag metainformációt ad hozzá a bytecode-hoz. Ez osztályok esetén lehet például, hogy melyik másik osztályból öröklődik, mezők és osztály függvények esetén lehet a láthatóság. A generikus osztályok és függvények is annotálva vannak a típus paramétereik számával. Ezeknek az információknak egy részét a runtime más részét a debugger használja fel.

1. A Parser

A Parser komponensen végzett munkám főként a default és named argumentumok parsolása.

* 1. A default argumentumok

A default argumentumokkal rendelkező függvények esetében nem csak egy, hanem kettő függvényt is elő kell állítani. Az eredeti függvény mellett a második egy wrapper függvény, aminek feladata, hogy runtime megvizsgálja, hogy ténylegesen mely argumentumok lettek átadva, és a hiányzó argumentumok default incializáló expressionjét meghívja.

Az első feladat létrehozni egy új AST node-ot. Ez az AST node az ir::*ETSParameterExpression* tartalmazza az információt minden függvény, minden argumentumáról. Gyakorlatilag minden *ir::ETSParameterExpression* két részből áll, amit az ETS nyelv nyelvtana diktál. Az egyik egy *ir::Identifier*, a másik pedig egy *ir::Expression*. Ezek közül a második az argumentum default initializer-e, ami opcionális. <kép a node-ról>.

A következő feladat a létrehozott AST node parsolása.

## Nyilatkozat

Alulírott Szűcs Dániel mérnökinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, mérnökinformatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

2023.

Szűcs Dániel