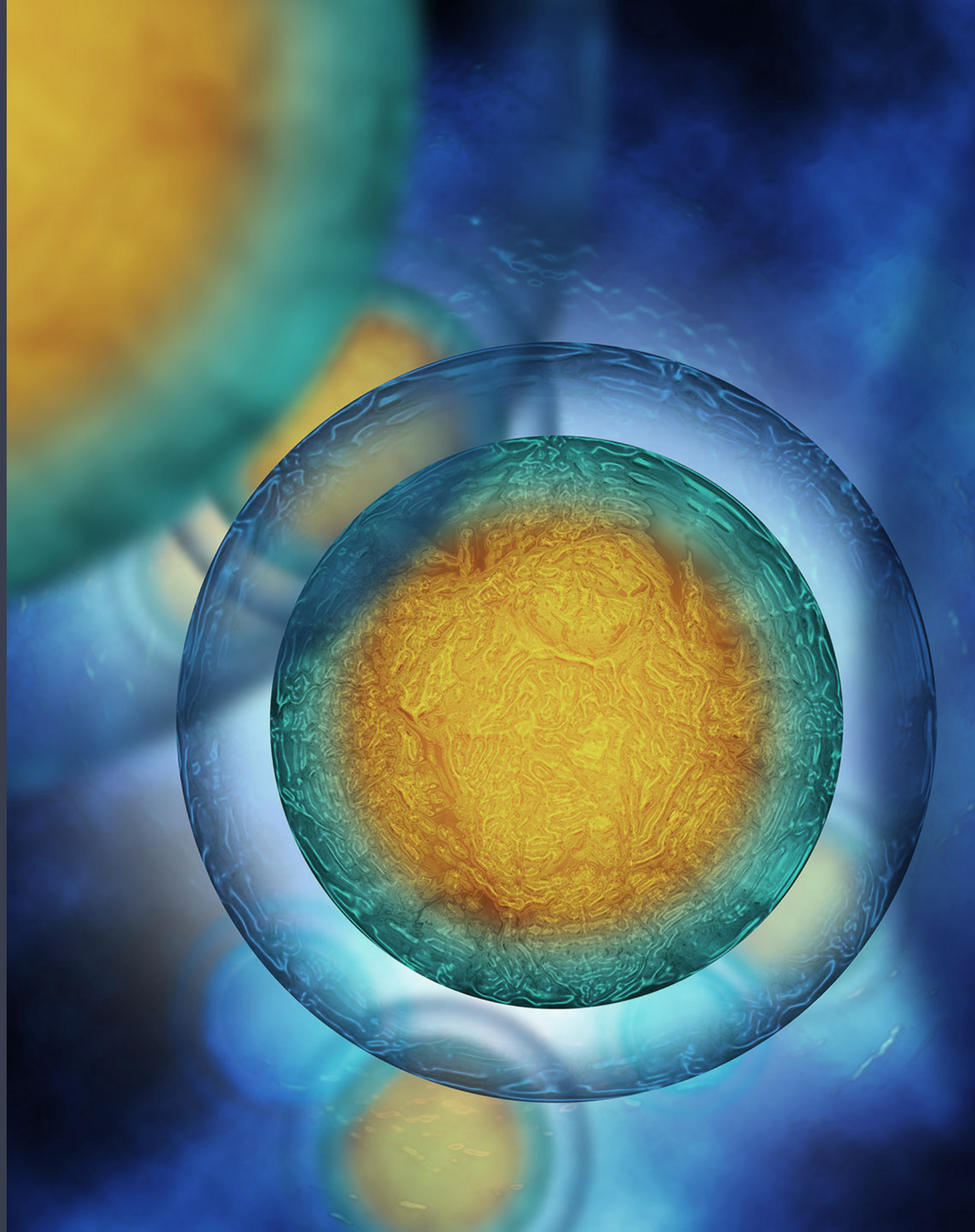
FE3CWS

# A HÁROM "CO" TÉLI ISKOLA TANANYAGA

A 2017-1-SK01-KA203-035402 számú ERASMUS+ projekt O1 megjelölésű intelektuális kimenete



# BEVEZETÉS

## gyanánt

- 9 téma szoftverfejlesztésről, -megértésről és helyességről
- 10 szerző Európa 7 egyeteméről, Horvátországból, Magyarországról, Hollandiából, Portugáliából és Szlovákiából
- Elérhető 7 nyelven: angolul, magyarul, szlovákul, horvátul, románul, bolgárul és portugálul

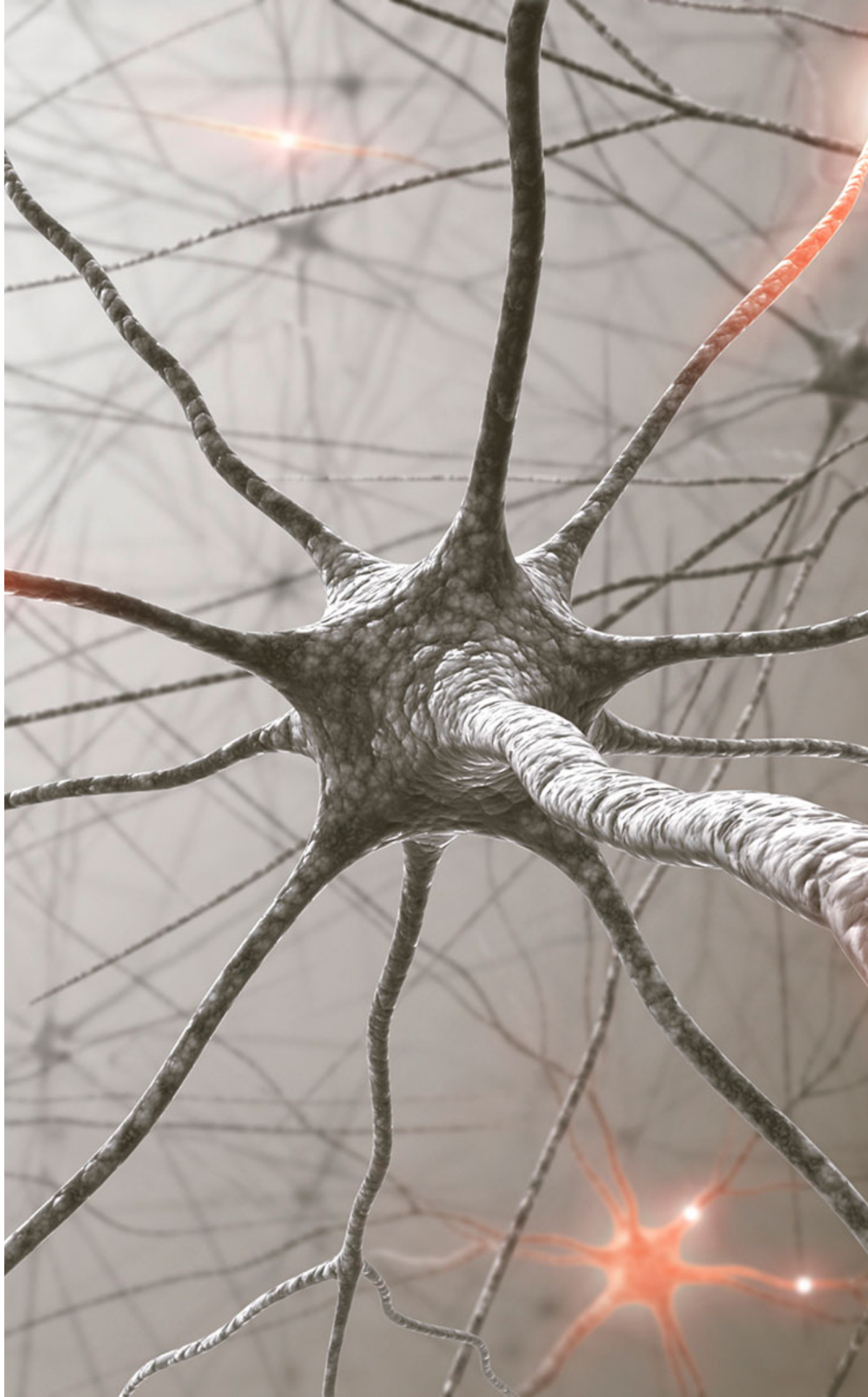
Co-funded by the Erasmus+ Programme of the European Union

A három “CO” (Composability, Comprehensibility and Correctness) téli iskola (3COWS) az első intenzív program felsőoktatási tanulók és tanárok számára a közép-európai funkcionális programozás (CEFP) nyári iskolája közösségének kiterjesztését célzó 2017-1-SK01-KA203-035402 számú “Focusing Education on Composability, Comprehensibility and Correctness of Working Software” ERASMUS+ projekt keretében, amelyet 2018. január 22. és 26. között került megrendezésre.

A mellékelt anyagot a fenti projekt keretében hozták létre és mutatták be. Ez a kiadvány a projekt szellemi kimenetének nyomtatott formátumú verziója.

# TARTALOMJEGYZÉK





# A TERMÉSZETES NYELV MEGÉR-TÉSE SZÁMÍTÓ-GÉPEKEN

A gondolkodó gép építése nagy kihívás. A nyelv és a tudat viszonya a számítógépes tudósok számára érdekes, tágabb értelemben a történészek, a pszichológusok, a nyelvészek és a filozófusok is foglalkoznak vele.

Például Renfrew [31] úgy jellemzi az embert mint szimbolikus lényt, figyelembe véve, hogy a természetes nyelvi fejlődés a szimbólumok cseréjének eredménye. Gardenfors [14] megállapítja, hogy a nyelvi fogalmak hierarchikus felépítésűek. A nyelvtan evolúciója [24] és a nyelvek genetikai fejlődése [15] hatással van a strukturális terjeszkedés problémájára, ezért ezek csak az egyszerű nyelvek számára alkalmazhatók determinisztikus módon. Kiderül, hogy a genetikai vonalak a szemantikai következtetések értékének szerepében sokkal inkább ígéretes megközelítést jelentenek egy nyelvi evolúcióhoz, mint az emberi agy genetikai struktúrájának egyenes alkalmazása. Azt is felismertük, hogy az evolúció folyamata különböző nyelvi meta-szinteken zajlik le.

Chomsky hipotézise az egyetemes nyelvtan létezésére minden természetes nyelv számára [7] nagyon érdekes és motiváló. De talán az univerzális nyelvtant nem szabad úgy értelmezni, mint egy fix és parametrizált grammatikai struktúrát, hanem inkább mint egy univerzális algoritmust, vagyis determinisztikus folyamatot egy meta-nyelvi szinten, amely az értelmes, azaz szemantikai értékeket ábrázoló paraméterekkel paraméterezhető. Edelman [12] kimondja, hogy a gépi tudatnak van nyelvi anyaga, ami szimbolikus és ugyanakkor értékalapú is. Ebből következik, hogy az elme nemcsak szimbolikus, hanem egy dinamikus folyamat is, amely a nyelv szüntelen változását engedélyezi. Végül pedig itt van a Shaumyan által alkalmazott univerzális nyelvtan [34], a nyelv dinamikus, sőt alkalmazási folyamatokban is kifejeződik.

A szemiotikai nyelvelmélet előnye, hogy a szintaxis és a szemantika egymással kölcsönösen kötődnek egymáshoz, és nem szétválaszthatók. Természetesen ez arra hívja fel a figyelmet, hogy szemantikai kategóriákat kell figyelembe venni a nyelvváltozás kifejezésére.

Véleményünk szerint a nyelvek szemiotikus elmélete lehetőséget ad a gépi elme determinisztikus evolúciójára a természetes nyelvek és formális nyelvek részhalmaza számára a különböző kommunikációs formákban. Ebben az értelemben a nyelveken alapuló ember-számítógép és számítógép-számítógép kommunikáció egységes lehet. Ennek az ötletnek a kiindulópontja a rendszeres nyelvekre korlátozódik [17]. Jelenleg ismerjük az algoritmust a nyelvi fogalmaknak a gépi elme belső nyelvére történő átalakulásához, és képesek vagyunk arra is, hogy ezen belső nyelvből fogalmakat hozzunk létre. Ez a belső nyelv az emberi gondolkodás belső nyelvének analógiája - egy csendes nyelv a hangos természetes nyelv mögött. Jelenleg nem ismerjük sem a konceptualizációs szabályokat, sem a fogalmi fogalmakra vonatkozó szabályokat a gépi elméletben. Mindazonáltal világos módszertanunk van a géptudomány evolúciójára, melyet nagymértékben párhuzamosan alkalmazott dinamikus folyamatok képviselnek, amelyek a nyelvi anyag nem redundáns információit képviselik. Ráadásul a nyelvi fogalmak absztrakciójának növelése csökkenti a meta-műveletek számát és növeli az alkalmazási kötések számát.

Kiindulunk a gyermeki gondolkodás hatékony algoritmusaiból, és az információáramlás sebességét 300 billió jelre becsüljük az emberi agyban másodpercenként. Ezután bemutatjuk a szlovák szöveges és fonetikus grammatikát, valamint az elemzett példaszöveget. Továbbá szemléltetjük a különböző absztrakció nyelvi elemeinek megszerzésének folyamatát, és a hierarchikus hash-táblákat mint belső modellt. A kiválasztott könyv teljes szövegének beszerzési folyamatát is értékeljük, és arra a következtetésre jutunk, hogy a nyelvszerzés hierarchikus absztrakcióval nem redundáns gráfot eredményez, ugyanolyan számú szekvenciális és párhuzamos összerendeléssel, mindezt anélkül, hogy fizikai adattárolásra lenne szükség.

Ezután grammatikai megközelítést alkalmazunk a kommunikált vizuális objektumok felismerésére. Először is ki kell derítenünk, hogyan írhatjuk le az objektumokat, majd alkalmazhatjuk az absztrakciót ezekre az adatokra is. Elsődlegesen a 3D-s objektumleírásra összpontosítunk a nyelvtanok segítségével. A nyelvtanelméletben ezt a lépést szimbolizációnak nevezik.

A szimbolizálás biztosítja az objektumleírást, és biztosítja az alapvető adatok absztrakt rétegét. Amint láthatjuk, az adatkivonás a funkcionális nyelv segítségével lehetővé teszi számunkra az objektumok elvonását és feldolgozását.

Alkalmazási megközelítés használható a nyelvfeldolgozásra akkor is, ha összefüggés nélküli nyelvtanokat használunk. Olyan algoritmust mutatunk be, amely képes bármely kontextusmentes grammatikát szuperkombinációs formára átalakítani. Az így kapott forma a bemeneti nyelvtan formájától függ, ezért új probléma merül fel: megtaláljuk a megfelelő nyelvtant a megfelelő feladathoz? Röviden bemutatjuk a különböző nyelvtípusok szuperkombinációs formáit, és összehasonlítjuk azok tulajdonságait. Az algoritmus hatékonyságát hasonlóképpen hasonlítjuk össze a bemutatott grammatikai esetekbe, és azt mutatjuk be, hogy az algoritmusunk javulhat, ha grammatikai ciklusokat használunk. Az eredményül kapott szuperkombinációs formákat az elemek nyelvtani tömörítésével és újbóli felhasználhatóságával is tárgyaljuk, amelyek a nagyobb léptékű szövegek mint bemeneti minták feldolgozásának végeredményei.

## Hivatkozások


Renfrew, C.: Prehistory: The Making of the Human Mind. Weidenfeld & Nicholson, (2009)

Gardenfors, P.: Symbolic, conceptual and subconceptual representations, Human and Machine Perception, pp. 255-270, (1997)

ONeill, M., and Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation, Vol. 5, No. 4, pp. 349–358, (2001)

Hugosson J., Hemberg E., Brabazon A., ONeill M.: Genotype Representations in Grammatical Evolution. Applied Soft Computing, Vol.10, No.1, pp.36-43, (2010)

Chomsky, N.: Syntactic Structures, Walter De Gruyter: Mouton classic, (1957)

Edelman, Sh.: Computing the Mind: How the Mind Really Works, (2008)

Shaumyan, S: A Semiotic Theory of Language. Bloomington: Indiana University Press, (1987)

Kollar, J.: Formal Processing of Informal Meaning by Abstract Interpretation, Smart Digital Futures 2014, June 18-20, Chania, Greece, pp. 122–131, (2014)

# STATIKUS KÓDELEMZÉS A CODECHECKER SEGÍTSÉGÉVEL

## ÁTTEKINTÉS

A szimbolikus végrehajtás [4] egy népszerű statikus elemzési módszer, amelyet mind a programellenőrzésben, mind a hibakeresési eszközökben használnak. Úgy működik, hogy értelmezi a kódot, bemutatva egy szimbólumot minden fordításidőben ismeretlen értékre (például a felhasználó által megadott bemenetekre), és szimbolikusan végzi a számításokat. Az elemzőmotor több végrehajtási útvonal vizsgálatára törekszik egyszerre, bár az összes útvonal ellenőrzése nehézségekbe ütközik a lehetőségek nagy számának köszönhetően.

Míg a programellenőrző eszközökhöz gazdag irodalom létezik, a hibakereső eszközöknél normál esetben meg kell elégednünk az egyes technikákra vonatkozó felmérési cikkekkel [1]. Ebben a tanulmányban nemcsak az egyes módszerekről beszélünk, hanem arról is, hogy ezek a döntések hogyan hatnak egymásra és erősítik egymást, létrehozva egy olyan rendszert, amely nagyobb, mint részei. A Clang Static Analyzer [2] nevű hibakereső eszközre (a továbbiakban: elemző) és egy körülötte épített infrastruktúrára összpontosítunk [3]. A hangsúly a végponttól végpontig terjedő skálázhatóság elérése.

Ez magában foglalja az elemzés futási idejét és memóriafogyasztását, a felhasználók hibabejelentését, a hamis pozitív eredmények automatikus elnyomását, az inkrementális elemzést, az eredmények eredményeit és a folyamatos integrációs hurkok használatát.

Megmutatjuk a jövőbeli irányokat, és megnyitjuk az ilyen eszközökre vonatkozó problémákat.

Bár az elemző csak a C / C ++ / Objective-C kódot képes kezelni, az itt bemutatott technikák nyelvfüggetlenek és más hasonló statikus elemzési eszközökre is alkalmazhatók.

# CLANG STATIC ANALYZER

Összefoglaljuk a szimbolikus végrehajtás és programvégrehajtás munkamódszereit az analizátorban. Megmagyarázzuk a memória ábrázolását [6], az értékek és a memóriahelyek közötti kötések kezelését, valamint azon specifikus állapotok ábrázolását, ahol az ellenőrzés egy olyan modult tartalmaz, amelyet az analizátornak egy meghatározott típusú hiba megtalálásához írtunk. Bevezetjük a szimbolikus számítások fogalmát is. Az elemző által használt ábrázolások döntő szerepet játszanak abban, hogy a nagyszabású szoftverelemzés életképes legyen.

Mivel az összes lehetséges végrehajtási útvonal megvizsgálása ésszerű időn belül nem lehetséges, be kell mutatni az elemzési költségvetés koncepcióját: egy adott időtartamra vonatkozó becslés, amely lehetővé tenné egy adott kóddarab elemzését. A cél az, hogy minél több hibát találjunk, alacsony hamis pozitív arány mellett.

Megmutatjuk, hogy az elemző az elemzés szempontjából érdekesebb útvonalakat prioritásként kezeli, és hogyan lehet hatékony módon kiküszöböli a nemkívánatos utakat a korlátok megoldásában [5]. Az elemző számos heurisztikát alkalmaz, amelyek automatikusan elnyomják a hamis pozitív jeleket.

Amikor egy hibát talál, akkor a megfelelő útvonal és a korlátozások hasznosak a probléma megértéséhez. Ugyanakkor nem célszerű bemutatni az összes információt a felhasználónak. Megmutatjuk, hogy az elemző törekszik a felhasználónak egy tömör, mégis betekintő hibajelentés bemutatására, amely minimalizálja a hiba javításának idejét.

# CODECHECKER

A statikus elemzés skálázhatóságát nemcsak a számítási erőforrások hatékony felhasználása, hanem az emberi erőforrások hatékony felhasználása szempontjából is meghatározzuk, mint a fejlesztői idő. A CodeChecker olyan eszköz, amely megkönnyíti az Analizátor és más hasonló statikus elemzési eszközök integrálását beépítési rendszerekbe és folyamatos integrációs hurkokba. Ez egy teljes körű hibamegoldó rendszer, amely nyomon követi az ilyen eszközök által talált hibákat.

Tekintettel a fejlesztői idő véges költségvetésére és a nagy szoftverekre vonatkozó ezer jelentésre, fontos, hogy először értékeljük a leginkább a befektetés megtérüléséről szóló jelentéseket.

A CodeChecker támogatja a differenciális elemzést, amely megakadályozza a fejlesztők új hibák bevezetését anélkül, hogy előzetesen meg kellett volna szüntetniük az összes korábbi jelentést.

# ÖSSZEGZÉS

Ebben a dokumentumban összefoglaljuk az összegyűjtött tapasztalatokat, miközben hozzájárulunk a legkorszerűbb Clang Static Analyzer és CodeChecker termékekhez. Reméljük, hogy hasznos forrás lesz mindenki számára, aki a statikus elemzési eszközökön dolgozik.

## Hivatkozások


[1] Baldoni, R., Coppa, E., Delia, D.C., Demetrescu, C. and Finocchi, I., 2018. A survey of symbolic execution techniques. ACM Computing Surveys (CSUR), 51(3), p.50.

[2] Clang Static Analyzer, https://clang-analyzer.llvm.org/. Last accessed 4 Nov 2018

[3] CodeChecker, https://github.com/Ericsson/codechecker. Last accessed 4 Nov 2018

[4] King, J.C., 1976. Symbolic execution and program testing. Communications of the ACM, 19(7), pp.385-394.

[5] Kovacs, R., Horvath, G., 2018. An Initial Prototype of Tiered Constraint Solving in the Clang Static Analyzer. Studia Universitatis Babes-Bolyai: Series Informatica, 63:(2) pp. 88-101.

[6] Xu, Z., Kremenek, T. and Zhang, J., 2010, October. A memory model for static analysis of C programs. In International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (pp. 535-548). Springer, Berlin, Heidelberg.

# PROGRAMOZÁS A VIRTUALIZÁLT HÁLÓZATI ERŐ-FORRÁSOK IRÁNYÍTÁSÁBAN ÉS ORKESTRÁ-CIÓJÁBAN

A hálózati funkciók virtualizációja (NFV) egy új paradigma a hálózatok építésének és üzemeltetésének megváltoztatására. A hálózati erőforrások virtualizációs rétegen keresztül történő szétválasztása szükségessé teszi az NFV menedzsment és orkestrációs (MANO) funkciók készítését. A különböző funkcionális blokkokon végrehajtott menedzsment funkciók összehangolására koncentrálunk, hogy az elosztott környezetben működő MANO funkciók megbízható működést érjenek el. A kihívásokat az Open Stack virtuális technológia gyakorlati példáján és a távközlési ipar által inspirált problémákon szemléltetjük.

## BEVEZETÉS

Ezeknek az előadásoknak az a célja, hogy bemutassák az olyan új koncepciókat, mint pl. a Network Functions Virtualization (NFV), amelyek jelenleg komplex szoftverrendszerek és hálózatok keretében valósulnak meg. Bemutatjuk az újonnan felmerülő kihívásokat, és bemutatjuk a tanulóknak, hogy hogyan kezeljék őket a programozási technikák az elosztott környezetben működő virtualizált hálózati erőforrások menedzsmentjeinek és orkestrációs funkcióinak összehangolására.

# KOMPLEX RENDSZEREK

Ezeknek az előadásoknak a témája a komplex rendszerek, különösen a bonyolult szoftverrendszerek és komplex hálózatok elmélete. Ezért az előadások az elmélet gyengéd bevezetésével indulnak, gondosan elhelyezve a megfontolt problémákat és kihívásokat a hálózatok és szoftverrendszerek jelenlegi fejlődésében. A virtualizáció egy paradigma, amelyet gyakran alkalmaznak komplex szoftverrendszerek kezelésében. Ez magában foglalja az új absztrakt réteg bevezetését, a rendszerréteg virtuális kiadását és funkcióit, ami megakadályozza a rendszerrétegek közötti függőség létrejöttét.

# IRÁNYÍTÁSI ÉS ORKESZTRÁCIÓS FUNKCIÓK

A telekommunikációs hálózatokban új paradigmát vezetnek be, a Network Functions Virtualization (NFV), amely a hálózati függvényt a fizikai hálózati erőforrásokból egy új virtualizációs rétegen keresztül különíti el [2]. Mindazonáltal ez szükségessé teszi az NFV irányítási és rendezési funkciók (MANO) készletének fejlesztését. E célból az Európai Távközlési Szabványügyi Intézetben (ETSI) külön munkacsoportot határoztak meg. A hálózati függvény virtualizációs menedzsment és orkestrációs architekturális keret definíciója megtalálható az [1]-ben. Ezen előadásokban a különböző funkcionális tömbökben végrehajtott irányítási és rendezési feladatokra összpontosítunk, annak érdekében, hogy az elosztott környezetben működő menedzsment és orkestrációs funkciók megbízható működést érjenek el.

# PÉLDÁK

Ezek a jegyzetek bemutatják a témát, azzal a céllal, hogy megmagyarázzák a problémákat és a megoldásuk során használt elveket, módszereket és technikákat. A kidolgozott példák és gyakorlatok szolgálják a hallgatókat mint tananyag, amelyből megtanulják használni a funkcionális programozást az NFV-t használó elosztott komplex rendszerek irányítási és rendezési funkcióinak hatékony és hatékony összehangolására.

Az ezen előadásokban kifejtett módszerek és technikák, valamint a hálózati virtualizáció menedzselésének és rendezésének problémái már léteznek, és ebben az értelemben nem eredetiek. Ezeknek a feljegyzéseknek az a célja, hogy ezeknek a módszereknek tananyagként szolgáljanak.

Az irányítás és az orkestrációs funkciók összehangolásának problémáit és kihívásait az OpenStack platform [3] használja. Ez egy nyílt forráskódú operációs rendszer, amely integrálja a szoftver modulok gyűjteményét, amelyek szükségesek a cloud computing réteges modell biztosításához. Az ilyen technológiák szükségesek az aktuális hálózatok virtualizációs paradigmájából eredő problémáinak kezelésében. Az OpenStack megoldásainak megértését elsajátító hallgatók képesek lesznek ismereteiket más meglévő technológiákkal azonos vagy hasonló célokra átvinni.

Az új hálózati paradigmákból és azok megoldásaiból fakadó kihívások gyakorlati példákkal illusztrálhatók az OpenStack virtuális technológiát alkalmazva, példaként a telekommunikációs iparág problémáit felhasználva.

# Hivatkozások

[1] ETSI Industry Specification Group (ISG) NFV: ETSI GS NFV- MAN 001 v1.1.1: Network Functions Virtualisation (NFV); Management and Orchestration European Telecommunications Standards Institute (ETSI), 2014, https://www.etsi.org/deliver/etsi_gs/NFV- MAN/001_099/001/01.01.01 60/gs_NFV-MAN001v010101p.pdf, accessed July 1, 2018

[2] Han, B., Gopalakrishnan, V., Ji, L., Lee, S.: Network function virtualization: Challenges and opportunities for innovations. IEEE Communications Magazine 53(2), 90–97 (2015)

[3] OpenStack Cloud Software. OpenStack Foundation (2018), www.openstack.org, accessed July 1, 2018

# FELHŐALAPÚ SZÁMÍTÁS-TECHNIKA ÉS FUNKCIONÁLIS PROGRAMOZÁS AZ OKTATÁSBAN

A felhőalapú számítástechnika kulcsfontosságú technológiává vált, és ezért sok számítástechnikai tananyag része. Alkalmazás-tervezés során a mikro-szolgáltatások kulcsfontosságúak. A mikrohálózatoknak tulajdonítható funkcionális bomlást szerver nélküli platformok, például az AWS Lambda szolgálhatják.

## MIKROSZOLGÁLTATÁSOK

Egyre több szakember javasolja egy mikrotervi architektúra elfogadását a felhő alkalmazások tervezésénél [1,2]. A mikroszolgáltató architektúrák közös jellemzője: az automatizált telepítés, az intelligens végpontok, a csendes komunikácó, az adatok decentralizált vezérlése [2]. A hagyományos felhőalkalmazások mikroszolgáltatói architektúrára való átállítása érdekében tett legutóbbi erőfeszítések figyelmeztetnek arra, hogy a sok, bár kicsi, kompozíciós szolgáltatás kezelésében egyre nagyobb a bonyolultság [3]. Itt fontos megkülönböztetni: a finomszemcsés felhőszolgáltatás üzleti logikai összetételére összpontosítunk az orkesztrációval. A tény, hogy a telepítésük orkesztrálva van-e (a végrehajtás központi eleme), vagy koreografálva van-e (minden egyes szolgáltatás önállóan működik), eldönti, hogy a kívánt alkalmazás szinkronvezérlést igényel-e vagy az aszinkron vezérlést. Mivel azonban az agilitás és a rugalmasság nagyon kívánatos tulajdonságok, a koreografált telepítés előnyösebb [1].

Azonban a mikroszolgáltatások nem funkcionális tulajdonságai, például a futásidejű teljesítmény, fontos szerepet játszanak az alkalmazás telepítésében. Ezért az AWS Lambda kontextusában szeretnénk megismertetni a diákokat a felhőben lévő teljesítményprofil fogalmával. Továbbá a kísérletek szabályozására kiegészítjük az AWS Lambda funkcionális paradigmáját egy Haskell-keretrendszer alkalmazásával.

# FELHASZNÁLÓKÖZPONTÚ FELHŐALAPÚ SZÁMÍTÁSTECHNIKA

A felhőalapú számítástechnika területén fontos szempont a felhasználók döntéseiben való segítségnyújtás. Ezek a döntések a következő kérdésekre vonatkoznak:

A. Hogyan működik az alkalmazás a virtualizált erőforrásokkal kapcsolatban?

B. Hány virtuális erőforrást és mely felhőalapú szolgáltatótól használjunk az alkalmazások telepítéséhez?

C. Meddig fogjuk használni? Mennyibe fog mindez kerülni?

Ezeket a kérdéseket rendszerint ütemezési problémaként modellezik, azon a feltevésen dolgozva, hogy az alkalmazásról nem áll rendelkezésre előzetes tudás. Egyszerű követelménycsomag, hogy az alkalmazás sikeresen telepíthető, és a költségek minimalizálhatók.

## A felhő alkalmazás ütemezőjének architektúrája

A BaTS-ütemtervet [4] pont azért fejlesztették ki, hogy segítséget nyújtson a felhasználóknak alkalmazásuk felhőbe telepítésénél. Ehhez önreprezentáló megközelítésre van szükség, és rendszeresen ellenőrizni kell a telepítés előrehaladását. Az első szakaszban a BaTS összegyűjti a statisztikákat.

Itt csak egy kis minta szükséges (30-50 feladat), hogy kiszámítsa a feladatok futási idejének átlagát és szórását különböző felhőalapú ajánlatokon. A költségvetési becslési modul ezután elvégzi a lineáris regressziót, hogy optimalizálja ezt a fázist.

## A BaTS módszertan használata könnyű virtualizációval

Bemutatjuk a hallgatóknak azt a problémát, hogy segítséget nyújtsanak az alkalmazástulajdonosoknak, akik a könnyű virtualizáció szempontjából a legmegfelelőbb választási lehetőségeket választanák mikrovállalkozásuk számára.

## AWS Lambda

Az AWS Lambda egy rendkívül könnyű, virtualizált számítási forrás, amelyet az Amazon kínál. A granularitás szintje és az ajánlott futásidő függvényenkénti híváskor legfeljebb másodpercekbe tellik. Az is feltétel, hogy nem lehet blokkoló a program viselkedése. 46 AWS Lambda típus létezik, eurónként kb. 1 GB * sec árazással.

## Az AWS API Haskell megvalósítása

Az AWS API [5] átfogó Haskell implementációja alapján célunk a BaTS becslési módszertan könnyű virtualizált erőforrásokra való alkalmazása.

## Gyakorlatok

Arra utasítjuk a diákokat, hogy a tényezők különböző lambda típusokon való futtatásának teljesítményét hozzákapcsolják a típushoz, majd lineáris regresszió segítségével létrehozzák az áteresztőképesség és az árhatékonyság görbét. Például meg kell fontolniuk, hogy melyik típus a legolcsóbb. Ezután azonosítaniuk kell, hogyan lehet hatékonyan megtalálni a leginkább nyereséges kombinációt: a legkisebb költség a legjobb teljesítmény érdekében.

## Az AWS Lambda mérése

Az AWS Lambda függvények tartályszerű virtualizált környezetben futnak. A funkciókhoz hozzárendelt számítási erőforrások mennyisége arányos a felhasználó által kért DRAM memóriával. Annak meghatározása érdekében, hogy a Lambda függvények miként tudják feldolgozni a különböző terheléseket, minden egyes számítási erőforrást egymástól függetlenül tudunk összehasonlítani. Ilyen mérések elvégezhetők a jól ismert számítási, memória- és hálózati intenzív terhelések példáin, mint az első N prímszám kiszámítása, a stream vagy iozone benchmark futtatása, vagy az adatok olvasása vagy írása az AWS S3-ra.

A kialakult módszertanok átadása kulcsfontosságú az oktatásban. A jövőbeli munkában szeretnénk támogatni a Haskell AWS Lambda funkciókat, amelyeket a Haskell AWS API megvalósításával telepítettünk.

# Hivatkozások


[1] Newman, Sam. Building Microservices. O'Reilly Media, Inc., 2015.

[2] Fowler, M., Lewis, J.: Microservices. http://martinfowler.com/articles/microservices.html (March 2014), Last accessed: 15-08-2018

[3] Balalaie, Armin, Abbas Heydarnoori, and Pooyan Jamshidi. "Migrating to cloud- native architectures using microservices: An experience report." arXiv preprint arXiv:1507.08217 (2015).

[4] AM Oprescu. Stochastic Approaches to Self-Adaptive Application Execution on Clouds. PhD Thesis, Amsterdam, Vrije Universiteit, 2013.

[5] https://hackage.haskell.org/package/amazonka-lambda-1.5.0, Last accessed: 15-08-2018.

# AZ ATTRIBÚTUM GRAMATIKÁK MODERN TÍPUSÚ, BIZTONSÁGOS BEÁGYAZÁSA

Az attribútum gramatikák egy erőteljes, deklaratív formalizmus, amely moduláris programok végrehajtását és formális indoklására hivatott. Bár egy teljes attribútum nyelvtani fordító egyedi igényekhez igazítható, végrehajtása rendkívül bonyolult, és hosszú távú karbantartásához nagyon nagy törekvés szükséges. Tény, hogy a hagyományos attribútum-nyelvtani rendszer fenntartása olyan nagy erőfeszítés, hogy a legtöbb a múltban javasolt rendszer már nem aktív. Az attribútum gramatikák modern megvalósításának megközelítése az, hogy a modern funkcionális programozási nyelv első osztályú elemeinek írjuk őket. Kijavítunk egy korábbi, cipzár alapú attribútum gramatika beágyazást, ami nem tolakodó (azaz nincs szükség változtatásokra a felhasználó által definiált adattípusokban) és típusbiztos. Ráadásul a modern Haskell kiterjesztésekkel egyértelműbb szintaxist érünk el. Hisszük, hogy beágyazásunk gyakorlatilag alkalmazható elegáns, hatékony és moduláris megoldások megvalósítására minden valós programozási kihívásra.

## BEVEZETÉS

Az attribútum gramatikák (AG) a Knuth [7] által a 60-as évek végén javasolt deklaratív formalizmus, amely lehetővé teszi, hogy a programokat modulárisan és kényelmesen megvalósítsák és korrektségüket indokolják. Egy konkrét AG egy kontextus-mentes grammatikára támaszkodik egy nyelv szintaxisának meghatározásánál, valamint a nyelvtan termeléséhez kapcsolódó attribútumokra, hogy meghatározza az adott nyelv szemantikáját.

Az AG-ket gyakorlatilag a valódi programozási nyelvek (pl. Haskell [2]), valamint a szép nyomtatási algoritmusok [16], az deforesztációs technikák [4] és az erőteljes típusrendszerek [11] meghatározására használják. Az AG-vel történő programozás esetén a modularitás megvalósulása miatt a számítások különböző szempontjainak meghatározása és használata különálló attribútumokként is lehetséges.

Az attribútumok különálló számítási egységek, amelyek jellemzően nagyon egyszerűek és modulárisak, és amelyek összetett programozási problémákhoz kidolgozott megoldásokká alakíthatók. Ezenkívül önállóan elemezhetők, hibázhatók és karbantarthatók, ami megkönnyíti a programfejlesztést és az evolúciót.

Az AG-k különösen hasznosak a fák számításainak meghatározásához: egy fa esetén több AG rendszer, mint például a [14,3,8,17], meghatározza, hogy mely értékeket vagy attribútumokat kell kiszámítani és elvégezni ezeket a számításokat. Az AG-rendszerek létrehozásához, fejlesztéséhez és karbantartásához szükséges tervezési és programozási erőfeszítések azonban óriásiak, ami gyakran leküzdhetetlen akadályt jelent a megérdemelt siker eléréséhez.

Az AG-k egyre népszerűbb alternatív megközelítése az általános célú programnyelvek első osztályú elemeinek [12, 9, 13, 15, 18, 1] beágyazására támaszkodik. Ezzel elkerülhető a teljesen új nyelv és a hozzá kapcsolódó rendszer megvalósításának terhe, a legmodernebb programozási nyelveken való elhelyezéssel. Ezt a megközelítést követve azután kihasználják az e nyelvek által már biztosított korszerű konstrukciókat és infrastruktúrákat, és a kifejlesztett tartományspecifikus nyelv sajátosságaira összpontosítanak.

A funkcionális cipzár [6] egy erőteljes absztrakció, amely nagymértékben leegyszerűsíti a helyi frissítések végrehajtásával járó átmeneti algoritmusokat. A funkcionális cipzárokat sikeresen alkalmazták egy AG beágyazására Haskellben [9,10]. Az elegancia ellenére ez a megoldás nagyon hátrányos volt, ami megakadályozta a valósidejű alkalmazások használatát: az attribútumok nem tárolódtak, hanem többször újraszámítódtak, ezzel súlyosan csökkentve a teljesítményt. A közelmúltban ezt a hibát megszüntették [5], de létrehoztak egy másikat: a megközelítést tolakodóvá tették, azaz a beágyazó felhasználó által definiált adatstruktúrák előnyeit közzel ki kell igazítani.

Ebben a cikkben egy alternatív mechanizmust mutatunk be az önszerveződő végtelen rácson alapuló attribútumok tárolására. Ez a fa a felhasználó által definiált algebrai adattípus (ADT) csúcsán helyezkedik el, és tükrözi annak struktúráját. A felhasznált ADT maga érintetlen marad. A beágyazás ezután két (náha egy) koherens cipzáron alapul, amelyek párhuzamosan haladnak az adatszerkezeteken. Nem tolakodó megoldásunk teljesen típusbiztos. A modern Haskell-kiterjesztések, mint például a ConstraintKinds lehetővé teszik számunkra, hogy propagáljuk a korlátozásokat lefelé az ADT-ben, és teljes mértékben kiküszöböljük a korábbi verziókban megjelenő futásidei típusváltásokat. A modern Haskell-funkciók egy másik előnye a tisztább szintaxis, mivel a Haskell sablon segítségével jóval kevesebb kódot generálunk.

# Hivatkozások


[1] Balestrieri, F.: The Productivity of Polymorphic Stream Equations and The Com- position of Circular Traversals. Ph.D. thesis, University of Nottingham (2015)

[2] Dijkstra, A., Fokker, J., Swierstra, S.D.: The architecture of the Utrecht Haskell compiler. In: Haskell Symposium. pp. 93–104 (2009)

[3] Dijkstra, A., Swierstra, D.: Typing Haskell with an Attribute Grammar (Part I). Tech. Rep. UU-CS-2004-037, Institute of Information and Computing Sciences, Utrecht University (2004)

[4] Fernandes, J.P., Saraiva, J.: Tools and Libraries to Model and Manipulate Circular Programs. In: Symposium on Partial Evaluation and Program Manipulation. pp. 102–111. ACM (2007)

[5] Fernandes, J.P., Martins, P., Pardo, A., Saraiva, J., Viera, M.: Memoized zipper- based attribute grammars and their higher order extension. Science of Computer Programming (2018)

[6] Huet, G.: The zipper. Journal of functional programming 7(5), 549–554 (1997)

[7] Knuth, D.: Semantics of Context-free Languages. Mathematical Systems Theory 2(2) (June 1968), Correction: Mathematical Systems Theory 5 (1), March 1971.

[8] Kuiper, M., Saraiva, J.: Lrc - A Generator for Incremental Language-Oriented Tools. In: International Conference on Compiler Construction. pp. 298–301. Springer-Verlag (1998)

[9] Martins, P., Fernandes, J.P., Saraiva, J.: Zipper-based attribute grammars and their extensions. In: Programming Languages - 17th Brazilian Symposium, SBLP 2013, Brasılia, Brazil, October 3 - 4, 2013. Proceedings. pp. 135–149 (2013)

[10] Martins, P., Fernandes, J.P., Saraiva, J., Wyk, E.V., Sloane, A.: Embedding at- tribute grammars and their extensions using functional zippers. Science of Computer Programming 132, 2 – 28 (2016), selected and extended papers from SBLP 2013

[11] Middelkoop, A., Dijkstra, A., Swierstra, S.D.: Iterative type inference with attribute grammars. In: International Conference on Generative Programming. pp. 43–52. ACM (2010)

[12] de Moor, O., Backhouse, K., Swierstra, D.: First-Class Attribute Grammars. In: 3rd. Workshop on Attribute Grammars and their Applications. pp. 1–20. Ponte de Lima, Portugal (2000)

[13] Norell, U., Gerdes, A.: Attribute Grammars in Erlang. In: Workshop on Erlang. pp. 1–12. 2015, ACM (2015)

[14] Reps, T., Teitelbaum, T.: The synthesizer generator. SIGPLAN Not. 19(5), 42–48 (Apr 1984)

[15] Sloane, A.M., Kats, L.C.L., Visser, E.: A pure object-oriented embedding of attribute grammars. Electronic Notes in Theoretical Computer Science 253(7), 205–219 (2010)

[16] Swierstra,D.,Azero,P.,Saraiva,J.:Designing and Implementing Combinator Languages. In: Third Summer School on Advanced Functional Programming. LNCS Tutorial, vol. 1608, pp. 150–206. Springer Verlag (1999)

[17] Van Wyk, E., Bodin, D., Gao, J., Krishnan, L.: Silver: an Extensible Attribute Grammar System. Electronic Notes in Theoretical Computer Science 203(2), 103–116 (2008)

[18] Viera, M., Swierstra, D., Swierstra, W.: Attribute Grammars Fly First-class: how to do Aspect Oriented Programming in Haskell. In: International Conference on Functional Programming. pp. 245–256. ACM (2009)

# MENNYIRE ZÖLD A FEJLESZTÉSI FOLYAMATA?

A Bio termékekhez hasonlóan a világ egyre inkább természetbarát ökoszisztémává válik. A zöld kezdeményezés két fő célt határoz meg: az energiafogyasztás csökkentése és a természetes energiaforrások használata az elektromosenergia-termelésben.

Az akkumulátorgyártók legnagyobb kihívása az, hogy mennyi ideig képes az akkumulátor folyamatosan működni anélkül, hogy újra fel lett volna töltve. Sok más kihívás is van, például a méret, amely nagymértékben befolyásolja a mobileszköz formáját és súlyát. Az akkumulátort kissé könnyebbnek kell tekinteni az olyan eszközhöz képest, amelyeknek az akkumulátorra van szüksége. A kihívás az, hogy hogyan méretezhetjük kisebbre és könnyebb súlyúra, hogy minden bizonnyal nagy hatékonyságot érjünk el a mobilkészülék működési idejének terén.

A hardveres kihívás csúcsán egy szoftveres kihívás is létezik: támogassa maga a szoftver az energia-megtakarítást. A felhasználói élmény korlátozása nélkül ma is úgy tekintik ezt a célt, mint egy elhallgatott, de fontos célt minden bármilyen hordozható eszközre irányuló [1] szoftverfejlesztésnél. Ez a probléma rendszerint akkor jelentkezik, amikor a kapcsolódó követelmények fontos szemponttá válnak.

Nem hagyhatjuk figyelmen kívül azt sem, hogy bármely mobil eszköz energiafogyasztását a futó alkalmazások befolyásolják (a szolgáltatások használati gyakoriságától a felhasználó tényleges hangulatáig), így az ilyen eszközökhöz való szoftverfejlesztés már most is kihívást jelent [4]. Azt is mondhatjuk, hogy a szoftverfejlesztési kihívás mindig ugyanaz, de a "mobilitást" kulcsfontosságú rendszertulajdonságként kell kiemelnünk. Az akkumulátor töltöttségi állapota is meghatározza a rendszer teljesítményét - ismert pl. az operációs rendszer "energiatakarékossági preferenciák" szintjének beállítása.

Még nehezebb feladat, ha egy tanár akarja felkészíteni a diákokat az ilyen kihívásokra [6]. Minden már ismert "legjobb gyakorlatot" és “energiatakarékossági tippet" olyan kontextusban kell bemutatni, amely könnyen érthető a hallgatók számára. Ezt úgy tehetjük meg, hogy a koncepciókat ismert környezetbe helyezzük, mint pl. szoftvertesztelés és teszt automatizálás [2]. E a következő szakaszok tartalma, kezdve a problémával, követve példákkal, majd további javításra vonatkozó tippekkel lezárva.

A fő hangsúly a működő szoftverek és a fejlesztési folyamatok energiafogyasztása, ahol minden fejlesztési fázis jelentős szerepet játszik. Tekintetbe véve a szoftverfejlesztési folyamatot, energiát fogyasztunk a problémaelemzés, a kód megalkotása és értékelése alatt is [3]. Szoftvereket vagy hardvereszközöket kell használni az energiafogyasztás-felügyelet végrehajtásához a kiválasztott operációs rendszereken futó szoftverekhez és az energiafogyasztás kiértékeléséhez egyaránt. A szokásos mérési forgatókönyvek a kiválasztott szoftverek energiafelhasználásának mérésére irányulnak[5], de megvizsgáljuk ezen eszközök használatának lehetőségét zöld a folyamatok mérésére is. A példáink számos helyzetet fednek le. A harmadik fél által kifejlesztett szoftverek energiaprofilozásának konkrét felhasználási forgatókönyveiből kiindulva kiemeljük a meglévő eszközök kulcsfontosságú tulajdonságait (előnyeit és hátrányait). A kifejlesztett szoftver tesztelése során bemutatjuk az energiaprofil-szoftverek tipikus használatát.

Az utolsó példánk a mérési megközelítés skálázhatóságára irányul egy kódrészlettől egy alkalmazás profilalkotásán keresztül az eszközláncok energiafogyasztási értékeléséig. Ez egy általános megközelítése az energiaprofilozásnak, és célja a kérdőjelek ponttá változtatása az általunk lefedett egyedi esetekre vonatkozó eredményértékelés alapján.

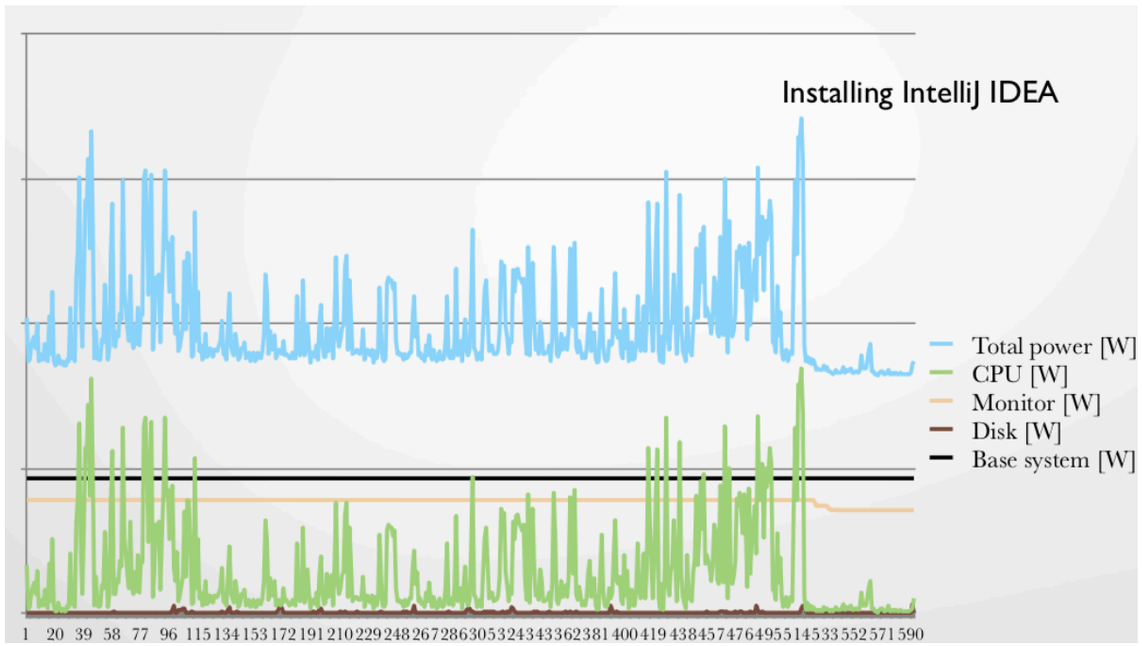
## Hivatkozások


[1] D. Li, W. G. J. Halfond, An investigation into energy-saving programming practices for android smartphone app development, in Proc. of the 3rd International Workshop on Green and Sustainable Software, GREENS 2014, ACM, 2014, pp. 46–53.

[2] D. Li, Y. Jin, C. Sahin, J. Clause, W. G. J. Halfond: Integrated energy-directed test suite optimization, in Proc. of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014, ACM, 2014, pp. 339–350.

## The energy-measured development game

1. Setup the environment
2. Start the energy monitor
3. Develop (think, code, test, fix) for 15 minutes
4. Have a 5 minutes break (stop energy usage monitoring, set up the next one, get a coffee)
5. Finish (for this time) if there is no further idea
6. Repeat (jump to label 2)
7. Analyse collected data (energy efficiency of your development process) inside the team





[3] M. Santos, J. Saraiva, Z. Porkolab, D. Krupp: Energy Consumption Measurement of C/C++ Programs Using Clang Tooling, in Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Appli- cations, Z. Budimac, ed., Belgrade, Serbia, 11-13.9.2017, Paper No. 15, 8 pages, also published online by CEUR Workshop Proceedings No. 1938 ISSN 1613-0073.

[4] J.Saraiva,M.Couto,Cs.Szabo,D.Novak:TowardsEnergy-AwareCodingPractices for Android, Acta Electrotechnica et Informatica, Vol. 18, No. 1, 2018, pp. 19–25. https://doi.org/10.15546/aeei-2018-0003

[5] Cs. Szabo, E.M.M. Alzeyani: Measuring Energy Efficiency of Selected Working Software, Studia Universitatis Babe¸s-Bolyai Informatica, Vol. 63, No. 1, 2018, pp. 5–16. https://doi.org/10.24193/subbi.2018.1.01

[6] Cs. Szabo, J. Saraiva: Focusing software engineering education on green application development, in Conference of Information Technology and Development of Education – ITRO 2017, Novi Sad, Serbia, pp. 165–169, ISBN 978-86-7672-302-7.

# A KÉSZÜLÉKEK FUNKCIONÁLIS PROGRAMOZÁSA

## MEGJEGYZÉS

Ez a cikk az RWDSL18 cikk [2] egy kiterjesztett verziója. A jelenlegi dokumentumban ugyanazt a DSL-t használjuk, csak néhány kisebb kiterjesztés és fejlesztés után. Az aktuális dokumentum arra fog összpontosítani, hogy az mTask DSL hogyan használható a készülékek programozására, valamint az mTask programokkal foglalkozó magas szintű szimulátort iTask programként tárgyalja.

## BEVEZETÉS

Számos eszköz ma már egyszerű mikroprocesszorral van felszerelve. Jellemző példák a termosztátok, izzók, elektromos aljzatok, tűzjelzők, ajtónyitók és így tovább. Amikor ezek az eszközök képesek kommunikálni egymással vagy valamilyen távoli számítógéppel, azt mondják, hogy részei a tárgyak internetének, az IoT-nek. Az ilyen készülékek mikroszámítógépei nagyon eleörhetőek és mindenütt jelen vannak. Az olyan drága eszközök, mint a nagyon összetett feladatokkal bíró autók és számítógépek teljes körűen beágyazott számítógépet és megfelelő szoftvereket tartalmaznak. A legtöbb kicsi és viszonylag olcsó IoT eszközhöz hasonló ilyen beágyazott számítógép vagy túl drága vagy túl sok energiát fogyaszt; egy egyszerű és olcsó mikroprocesszor használható a szoftver futtatásához. Ezek a rendszerek nagyon korlátozottak számítási teljesítmény és memória terön, jellemzően 30 KB-4 MB flash memória áll rendelkezésre a program tárolásához. A memória élettartama 1000 írási ciklusra korlátozódik. A változók, a halom és a verem tárolására a rendszerek 2-40 KB RAM-mal rendelkeznek.

A processzor sebessége és a memória korlátai kizárják az operációs rendszer használatát. A készülék csak végrehajtja a készüléket vezérlő programot. Még az IoT eszközök vezérlőprogramjai is több feladatot tartalmaznak. Például, egy gombnyomásra egy másodpercig nyomon követheti a kijelzőt, másodpercenként frissítheti a kijelzőt, mérni tudja a hőmérsékletet pl. kétszer percenként, és legalább öt perc múlva bekapcsolhatja a fűtést, hacsak a gombot valaki nem nyomja meg előbb. A különböző időkeretek és ezek függései miatt a vezérlőprogram hajlamos arra, hogy rendezetlen legyen - függetlenül a felhasznált programozási nyelvtől. Ezenkívül az IoT eszközök különálló programokat futtatnak az alkalmazás többi részére a tárgyak internetén, és számos protokoll használatával kommunikálnak. Ezáltal a tárgyak internetésnek fejlesztése és karbantartása összetett és hibákat szül.

A feladatorientált programozás, a TOP, olyan könnyű szálakat kínál, amelyek könnyen összetett feladatokhoz köthetők. A feladatokat lépésről lépésre értékelik, és az ilyen lépések után más feladatok aktuális értékét is megvizsgálhatják. A TOP-ot először az iTask rendszerben [4,5] ágyaztuk be a Clean-be [6]. Az iTask rendszerben a primitív feladatok az automatikusan generált webes űrlapon keresztül gyűjtenek adatokat, vagy adatokat gyűjtenek más programokból és adattárolókból. A kombinátorokat a feladatok összetettebb feladatokhoz való hozzárendelésére használják. Ebben a cikkben megmutatjuk, hogy a TOP nagyon alkalmas az IoT eszközök programozására. A primitív feladatok a bemeneteket és az érzékelők aktuális értékét biztosítják. Az iTask rendszerhez nagyon hasonló konstruktorokat használunk a feladatok összetett feladatokhoz való kombinálásához. Az IoT eszközök általában lazán összefüggő feladatokat látnak el, amelyekkel vezérlik az érzékelőket, a működtetőket és az eszközök kommunikációját. A TOP stílusban történő programozás tömör programokat kínál. Ezeknek a feladatoknak a végrehajtása a kis mikrokontrollerek korlátain belül, nagyon korlátozott feldolgozási sebessögnöl és kevés KB memóriákkal említést érdemel. A felhasznált mikrovezérlők súlyos korlátai miatt az iTask rendszert nem lehet az IoT eszközökhöz csatlakoztatni, mivel egy tipikus iTask programhoz kb. 100 MB memória szükséges. Meghatározunk egy beépített domain-specifikus nyelvet, az eDSL-t, mTask néven az IoT eszközök számára. Ez az eDSL be van ágyazva az iTask rendszerbe, mivel azt tervezzük, hogy ezeket a TOP nyelveket teljes mértékben átjárhatóvá tesszük.

Ebből a tanulmányból megtudjuk:

- Bemutatunk az IoT eszközök számára egy feladat alapú funkcionális programozási nyelvet. A mikroprocesszoros programozáshoz használt korábbi nyelvünkhöz [3] képest az imperatív perifériás vezérlést referenciális transzparens konstrukciók váltják fel.

- Megmutatjuk, hogyan lehet funkcionális, kiterjeszthető, több nézetű, típusbiztos, beágyazott DSL-t létrehozni. Ez egy címke nélküli eDSL [1] lesz.

- A generált kód kis és lassú eszközökön, valamint nagyobb gépeken és szimulátorokban is fut.

- Az Arduino C ++ közbenső nyelvként való használata miatt ez a funkcionális eDSL sok különböző mikrokontrolleren fut.

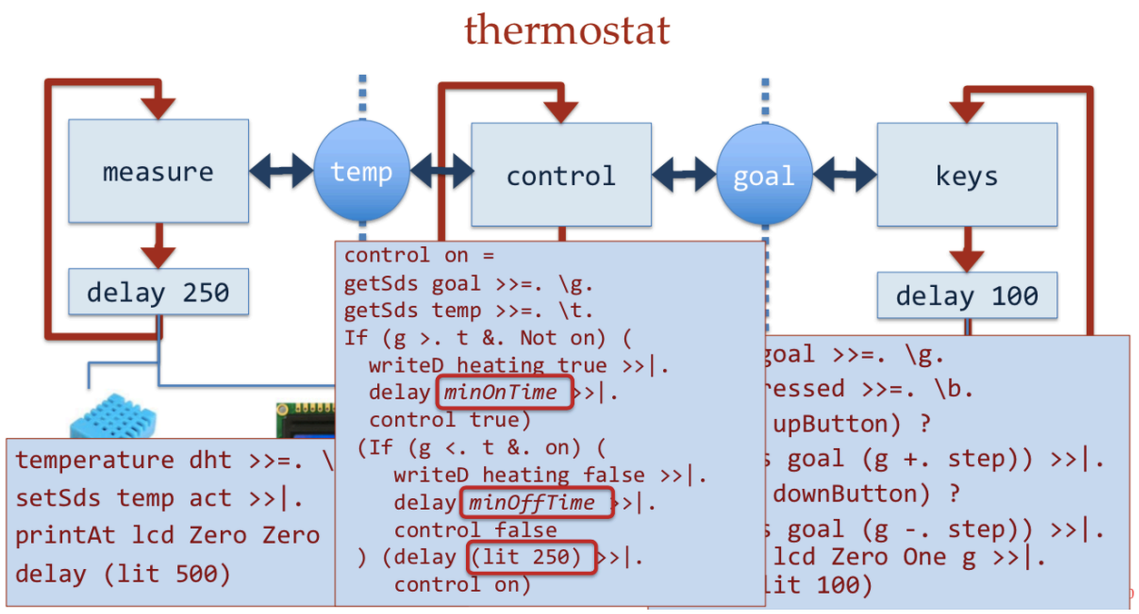
- Az mTask programok nagyszintű szimulációja egy iTask programban lehetővé teszi az eDSL-program hatásának megtekintését és a szimulált környezet manipulálását a meghatározott viselkedés kikísérletezéséhez. Egy ilyen szimulátorban sokkal könnyebb az idő és az érzékelők manipulálása, mint egy valós életbeállításban, pl. egy gombnyomással megváltoztathatjuk az érzékelő által jelzett hőmérsékletet, ahelyett, hogy fizikailag kitennénk az IoT eszközt ezeknek a hőmérsékleteknek.

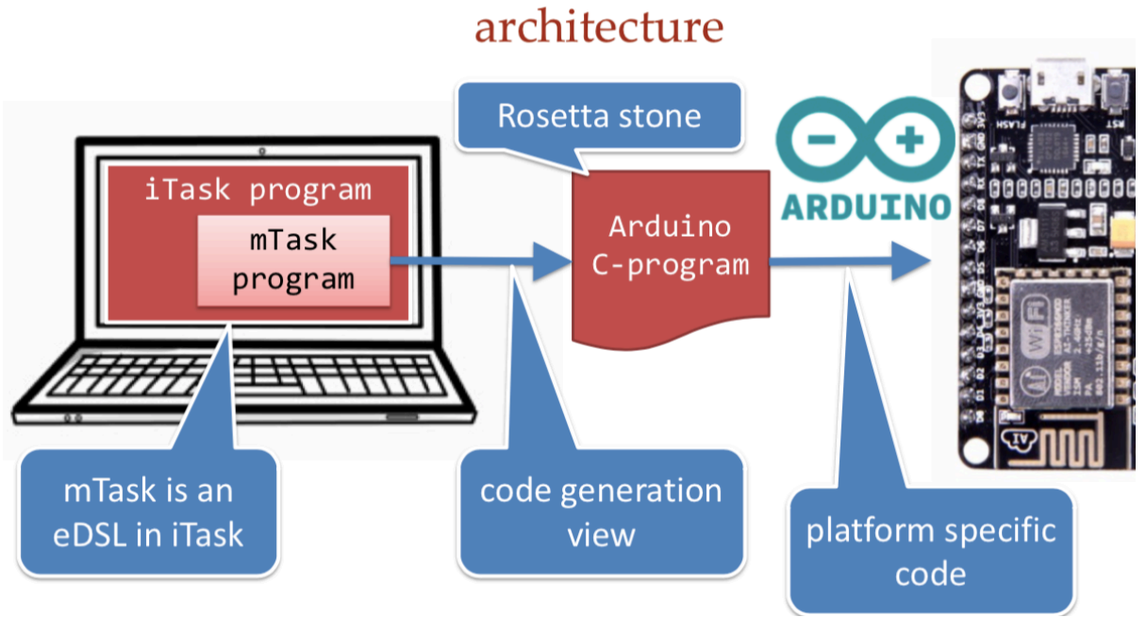
# Hivatkozások


[1] Carette, J., Kiselyov, O., Shan, C.c.: Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages. J. Funct. Program. 19(5) (Sep 2009). https://doi.org/10.1017/S0956796809007205

[2] Koopman, P., Lubbers, M., Plasmeijer, R.: A task-based dsl for microcomputers. In: Proceedings of the Real World Domain Specific Languages Workshop 2018. pp. 4:1–4:11. RWDSL2018, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3183895.3183902, http://doi.acm.org/10.1145/ 3183895.3183902

[3] Koopman, P., Plasmeijer, R.: A shallow embedded type safe extendable dsl for the arduino. In: Revised Selected Papers of the 16th International Symposium on Trends in Functional Programming - Volume 9547. pp. 104–123. TFP 2015, Springer-Verlag New York, Inc., New York, NY, USA (2016), http://dx.doi.org/ 10.1007/978-3-319-39110-6_6




[4] Plasmeijer, R., Achten, P., Koopman, P.: iTasks: executable specifications of inter- active work flow systems for the web. In: Hinze, R., Ramsey, N. (eds.) Proceedings of the ICFP'07. ACM, Freiburg, Germany (2007)

[5] Plasmeijer, R., Lijnse, B., Michels, S., Achten, P., Koopman, P.: Task-oriented programming in a pure functional language. In: Proceedings of the 14th Symposium on Principles and Practice of Declarative Programming. pp. 195–206. PPDP '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2370776.2370801, http://doi.acm.org/10.1145/2370776.2370801

[6] Plasmeijer, R., van Eekelen, M., van Groningen, J.: Clean language report (version 2.2) (2011), http://clean.cs.ru.nl/Documentation

# KÓDMEGÉRTÉS A CODECOMPASS-SZAL

## ÁTTEKINTÉS

A fejlesztés és a karbantartás két különböző fokozat, különböző jellemzőkkel rendelkezik, ezért különböző ezköztámogatást igényelnek. A fejlesztés során főként olyan új kódot írunk, amelyhez olyan támogatásra van szükség, mint például a kódok kitöltése, a zárójelek illesztése stb., És általában csak néhány fájllal dolgozunk, amelyek többé-kevésbé ugyanazon absztrakciós szintre vonatkoznak. A karbantartás során elsősorban a meglévő kódbázis olvasását és navigálását végezzük a különböző absztrakciós szinteken lévő modulok és fájlok nagy számán keresztül [1]. A fejlesztés során a szándékok világosak, ellentétben a kód megértésével, ahol a feladat az egyes kódtöredékek eredeti céljának visszaállítása.

Ipari környezetben [2] a projekt több millió sorból állhat. Hosszan meglévő nagy rendszerek esetében, ahol a kódbázisokat évtizedek óta fejlesztik és tartják fenn gyakran csapatcserével, az eredeti szándékok elvesztek, a dokumentáció megbízhatatlan vagy hiányzik, az egyetlen megbízható információ maga a kód. Az ilyen nagy szoftverrendszerek megértése alapvető, de általában nagyon nehéz feladat. Ez azt jelenti, hogy eszközi támogatására van szükség [3].

Ha ismeretlen forráskódot ismernénk meg, akkor az első lépés a rendszer megfelelő részeinek megkeresése. Ez a folyamat gyors azonosítást igényel, amely a naplóüzenetekből vagy más erőforrásokból beszerzett egyes entitások alapján történik. A következő lépés a rendszerrel kapcsolatos tudásunk kiterjesztése diagramokon, függvényhívási láncokon keresztül stb. Végül szeretnénk ellenőrizni az összegyűjtött ismereteket a verzióellenőrzési üzenetek, az architektúra információk és a kapcsolódó modulok hivatkozásain keresztül.

# CODECOMPASS

A CodeCompass [4, 5] egy nyílt forráskódú megértési keret. Olyan plugin architektúrát biztosít, amely lehetővé teszi a különféle vizualizációkat, információgyűjtőket, pointereket [6] stb. Elemző eszközök hozzáadását. A legfontosabb tervezési cél a CodeCompass alaklmazása nagyméretű ipari projekteknél.

Első lépésben a terméket meg kell vizsgálni: az összes információt összegyűjti és tárolja egy adatbázisban, amely ezután lehetővé teszi a szerverréteg számára a szükséges vizualizációkat. A gyors kereséshez a CodeCompass olyan szöveges indexelést használ, amely a forráskódban nyelvi független navigációt eredményez. Mivel az elsődleges cél az, hogy pontos információt adjon a nyelvi elemekről, a szimbólumok azonosítása a nevük szerint nem elegendő. Az LLVM fordítói infrastruktúrát a szimbólumok pontosabb azonosítására használjuk fel, valamint az elnevezett entitások feloldására az absztrakt szintaxisfa segítségével. A CodeCompass nyelvi elemzői bővülhetnek. A leginkább támogatott nyelvek a C / C ++, de részben kezelhető a Java és a Python is.

A megnevezett szimbólumokon kívül az adatbázisban további információk is tárolódnak, például az AST csomópontok (függvényhívások, öröklés) és a fájlok (interfész-szolgáltató kapcsolat, befogadás stb.) közötti kapcsolatok. Ezek a szimbólumok használatán alapuló architektikus szintű kép megjelenítésére szolgálnak [7].

A kódbázis nem az egyetlen forrása a dokumentációknak. A verzióellenõrzõ rendszer elkövetõ üzenetei olyan információkat is tartalmaznak, amelyek fontosak annak megértéséhez, hogy miért történtek bizonyos változások az adott modulban.

CodeCompass olvassa a Git adattárat is, ha van ilyen. A CodeCompass fejlett funkciókkal is rendelkezik. Megjelenítheti a fordító által generált funkciókat, amelyek hiányoznak a forrásból. A pointerelemzés segít megérteni, hogy mely változók hivatkoznak ugyanarra a tárgyra. A függvényhívási kapcsolatokat akkor is megvizsgálhatjuk, ha virtuális függvény vagy függvény pointer segítségével hívják fel őket.

# ÖSSZEGZÉS

A kód megértéséhez speciális eszköztámogatás szükséges, főleg a nagyméretű szoftverek megértéséhez. Továbbá szükséges a kódfelismerési eszközök áttekintése és kategorizálása az architektúrán és a funkcionalitáson belül annak érdekében, hogy megvizsgálhassuk képességeiket.

Bemutattuk a CodeCompass-t, amely a funkcionalitások széles skáláját mutatja be a vizualizációkkal, információszolgáltatással, verziókezeléssel és dokumentációgyűjtéssel, pointerekkl stb.

## Hivatkozások


[1] Jonathan Sillito, Gail C. Murphy, Kris De Volder. (2008). Asking and Answering Questions during a Programming Change Task. IEEE Transactions on Software Engineering, VOL. 34, NO. 4, July/August 2008.

[2] Porkolab, Zoltan & Brunner, Tibor & Krupp, Daniel & Csordas, Marton. (2018). Codecompass: an open software comprehension framework for industrial usage. 361- 369. 10.1145/3196321.3197546.

[3] Nathan Hawes, Stuart Marshall, Craig Anslow. (2015). CodeSurveyor: Mapping LargeScale Software to Aid in Code Comprehension. 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT) , 27-28 Sept. 2015.

[4] Porkolab,Zoltan & Brunner,Tibor (2018). The codecompass comprehension framework. 393-396. 10.1145/3196321.3196352

[5] CodeCompass, https://github.com/Ericsson/CodeCompass. Last accessed 5 Nov 2018.

[6] Brunner, Tibor & Porkolab, Zoltan. (2017). Two Dimensional Visualization of Soft- ware Metrics. Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications.

[7] B. De Alwis and G.C. Murphy. (1998). Using Visual Momentum to Explain Dis- orientation in the Eclipse IDE. Proc. IEEE Symp. Visual Languages and Human Centric Computing, pp. 51-54, 2006.

# FUNKCIONÁLIS PROGRAMOZÁSI SABLONOK A NAGYTELJESÍTMÉNYŰ SZÁMÍTÁSOK SZÁMÁRA

A funkcionális programozási nyelvek eszközöket biztosítanak az elosztott alkalmazások tervezéséhez és megvalósításához. Mivel a funkcionális nyelvek természetüknél fogva párhuzamos jellegűek, e tulajdonságukat ki lehet használni magasszintű elosztott és együttműködő megbízható párhuzamos feldolgozások számára. A jelenlegi párthuzamos szoftverek fejlesztése különböző módszereket és megközelítéseket alkalmaz a magas feldolgozási sebesség elérése érdekében. Mindemellett a párhuzamos alkalmazások fejlesztése az egyik legnehezebb feladatcsoportba tartozik, különösen funkcionális programozási megközelítés esetében.

A legfontosabb cél a párhuzamos programozási sablonok felfedezése egy új környezetben, melyel bemutathatjuk a funkcionális programozás alkalmazhatóságát egy új elosztott felállás esetén. Az ismert párhuzamos algoritmus sablonok egy körét teszteljük nagysebességű számítási környezetben. A példák szignifikáns része a sebesség növekedését mutatja. Az elérhető párhuzamosság mindig több összetevőtől függ, mint: az alkalmazott számítási sablon, a felbontás finomsága , az elosztott csomópontok szemantikája és az adatáramlás.  A példák mind a jól modellezett koordináció, mind a szemantikai helyesség kérdését vizsgálják.

# A SABLONOK ALKALMAZÁSI TERÜLETEI

## Koordináció

Az elosztott és párhuzamos funkcionális számítások speciális területe koordinációs nyelvi elemeket igényel. A korábbi kutatások azt vizsgálták, hogyan lehet a funkcionális nyelvek párhuzamos viselkedését és komunikációját magas szinten elérni. A bevezetett magas absztrakciós szintű funkcionális nyelvi elemek megmutatták, hogy lehetséges a párhuzamos magasszintű adatintenzív számítás [2].

Az ezt követő irányok a nyelviu elemek kifejezőerejének növelése irányába hatottak. Konkrétan a DClean, mint a Clean funkcionális nyelv koordinációs és elosztott célú nyelvi kiterjesztését eredményezték. A kiterjesztés magaszszintű nyelvi elemeket tartalmaz a tisztán funkcionális számítási csomópontok koordinálására az erre a célra alkalmazott elosztott klasztereken.

A konstrukció számítási dobozokat generál, melyek bufferelt adatcsatornákkal vannak összekötve. A DCLean használatával a fejlesztők jelezhetik, hogy az elosztott programozási minta hogyan szerveződjön egy generált elosztott gráfban és típusos csatornákon keresztül vezérlik a hálózati folyamatok adatfolyamait.A nyelv az elosztott és funkcionális alkalmazások írásának előnyeit adja anélkül hogy törődni kéne a többrétegű környezet és a közbülső szolgáltatások technikai részleteivel. A feladatok elosztása az előre definiált párhuzamos számítási sablonok, az algoritmusos sablonok, parametrizált függvények és típusok, valamint input folyamok segítségével történik.

A koordinációs nyelv legfontosabb célja, hogy funkcionális párhuzamos sémákat definiáljunk velük. Nagyszámú példa esetében a párhuzamosság komoly sebességnövekedést eredményezett. A ténylegesen elért párhuzamossága csatornák létrehozási sorrendjétől, az általuk végzett munkától és az adatok olvasási és írási sebességétől, valamint a csomópont komplexitásától függött [2].

Az elosztott számítások végrehajtható szemantikán alapuló megértést támogató eszköz segítségével támogatott grafikus megjelenítése [4] megkerülhetetlen volt az igazi elosztott rendszereknél.

Ez az eszköz jelezte a várt párhuzamosságot a jóldefiniált magaszintú sablonok által generált dobozok és csatornák esetében. Célja a DClean működési szemantikájának modellezése volt.

## Kiber-fizikai rendszerek

A sablon alapú funkcionális modellezés megközelítése, melyet a koordinációs nyelvek alkalmaznak alkalmazhatóak a ma divatos kiber-fizikai rendszerek (CPS) prototípusaira is. A CPS rendszerek és az elosztott rendszerek vagy a beágyazott rendszerek kapcsolatát vizsgálata fontos, hogy megfelelően megalapozott döntéseket hozzunk a kifinomult CPS rendszerek prototípusai esetében.

A CPS rendszer esettanulmányok megvalósítása [5] leírja a fizikai eszközöket (szenzorokat) vezérlő együttműködő számítási egységeket. és kapcsolatukat más komplex rendszerekkel. Az okosotthon CPS rendszer új aspektusokat, eszközöket és megközelítéséket alkalmaz a sablonok által. Az ilyen CPS rendszerek tervezése fontos szemantikai kérdéseket vet fel a valószínűségi és viselkedési szempontokból, ahol a vegyes működés az egyik fő elemzési és specifikációs kérdés volt.

# SABLONOK A NAGYSEBESSÉGŰ SZÁMÍTÁSOKHOZ

A sablonok használatának nagysebességű számításokra történő kiterjesztésének kutatása kulcs szempont a párhuzamos funkcionális megközelítés esetében. A korábbi ismeretek adaptálása a heterogén sokmagvas rendszerek sablon alapú programozására magasabb sebességet jelent. E kutatások mérések és összehasonlítások alapján elemzik az új párhuzamosítási törekvéseket.

A sablon prototípusok a funkcionalitásuk és koordinációjuk alapján lettek megadva. Az esettanulmányok illusztrálják a kapcsolatot más elvű elosztott rendszerekkel, ami a többrétegű programszerkezet szempontból fontos. A végrehajtható módon megadott elosztott rendszer jellemzők klaszterek és gridek funkcionális és elosztott sablonjai segítségével lettek letesztelve.

# Hivatkozások


[1] Zsok V.: D-Clean Semantics for Generating Distributed Computation Nodes, Work- shop on Generative Technologies, WGT 2010, Satellite workshop at ETAPS 2010, Paphos, Cyprus, March 27, 2010, pp. 77–84.

[2] Zsok V., Hernyak Z., and Horvath, Z.: Designing Distributed Computational Skeletons in D-Clean and D-Box. Central European Functional Programming School CEFP 2005, First Summer School, Budapest, Hungary, July 4-15, 2005, Revised Selected Lectures, LNCS Vol. 4164, Springer-Verlag, 2006, pp. 223–256.

[3] Zsok V., Koopman, P., Plasmeijer, R.: Generic Executable Semantics for D-Clean, Proceedings of the Third Workshop on Generative Technologies, WGT 2011, ETAPS 2011, Saarbrucken, Germany, March 27, 2011, ENTCS Vol. 279, Issue 3, Elsevier, December 2011, pp. 85–95.

[4] Zsok V., Porkolab Z.: Rapid Prototyping for Distributed D-Clean using C++ Tem- plates, Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae, Sectio Computatorica, Eotvos Lorand University, Budapest, Hungary, 2012, Vol. 37, pp. 19–46.

[5] Zsok V. et al.: Modeling CPS Systems using Functional Programming, Proc. of IFL17, Uni. of Bristol, pp. 168–174.