Forráskód feldolgozást végző alkalmazások fejlesztése és helyességvizsgálata

Tejfel Máté

Témavázlat

Forráskód feldolgozást végző alkalmazások

- Fordítóprogramok
- Refaktoráló eszközök
- Optimalizáló eszközök
- Elemző eszközök

Tartalom

Új programkönyvtár létrehozása

Haskell kibővítése

- Új nyelvi elem - Reference
- Adatelérés összetett struktúrákban
- Monádokat alkalmazó kifejezésekben is használható
- Automatikusan generálható adott típusokhoz

Reference könyvtár

Adatelérési absztrakció

- refSet
- refGet
- refUpdate
- axiómák a műveletekre

Reference könyvtár

Kombinálhatóak

- kompozíció — (_1 & just)
- additív kombinátor — (_1 &+& _2)
- disztributivitás
  - r & (p &+& q) ≡ (r & p) &+& (r & q)
  - (r &+& p) & q ≡ (r & q) &+& (p & q)

Beágyazott nyelvek

- sekély beágyazás
- mély beágyazás
- Haskell mint gazdanyelv

Miller

- magas szintű nyelv
- hatékony ("gyors") programok
- többszintű memóriahierarchia (scratchpad)
- nincs függvényhívás
- hívási verem helyett folytatás átadás
- adatstruktúra konfigurálás
- optimalizált assembly kimenet
  - MIPS32
  - "ipari változat"

Feldspar

Functional **E**mbedded **L**anguage for **DS**P and **PAR**allelism

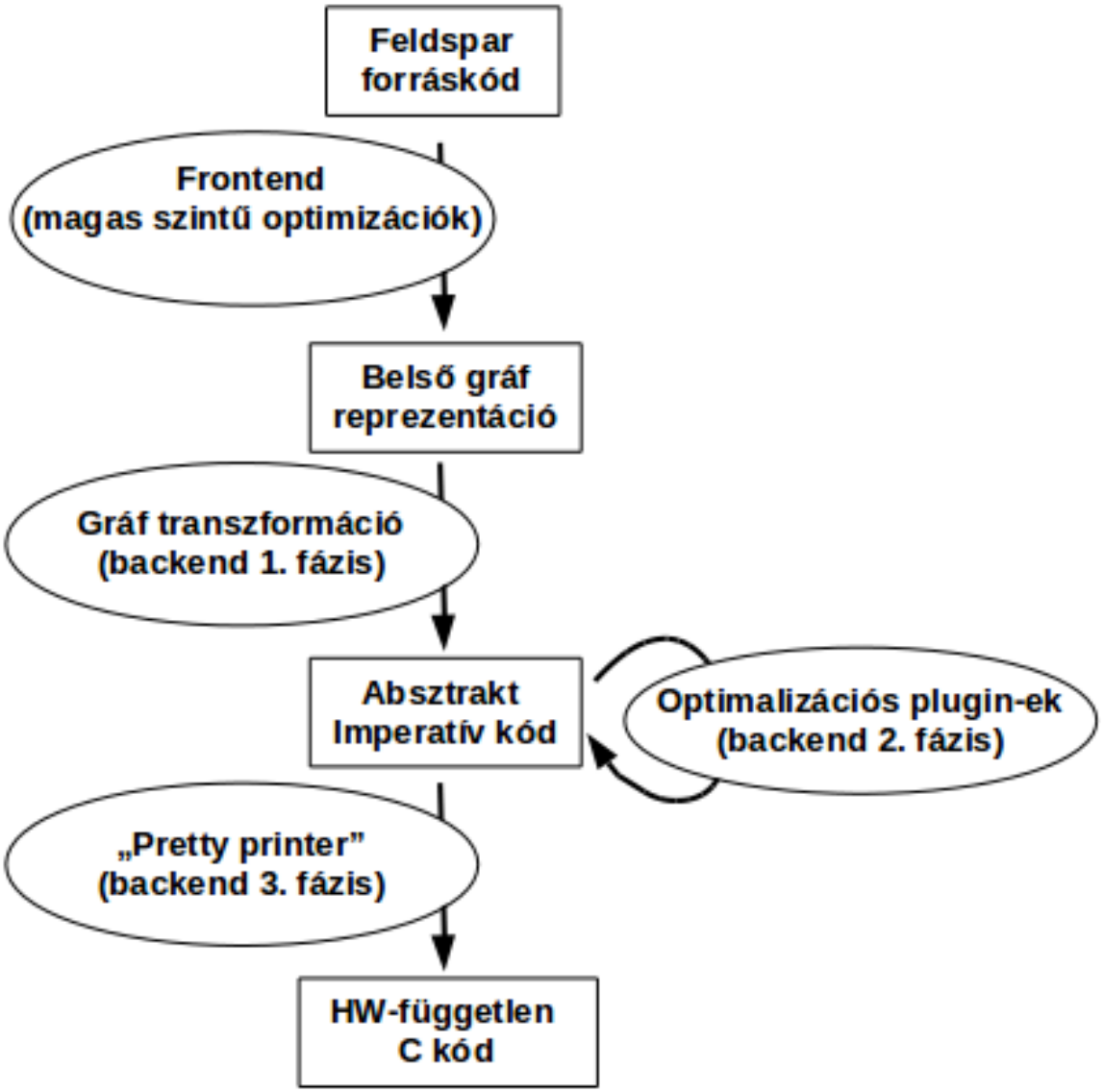
- digitális jelfeldolgozó algoritmusok
- magas szintű, deklaratív nyelv
- Kutatási együttműködés
  - Ericsson
  - Chalmers University of Technology (Göteborg, Svédország)
  - ELTE

Feldspar

- `Vector` típus, indexezett sorozat
- optimalizált C kimenet
- kézzel írt (hardverfüggetlen) C kóddal összevethető teljesítményű
- általános, optimalizációs keretrendszer
- hardverfüggő optimalizációk lehetősége

```
sumSq :: Data Int -> Data Int
sumSq n = sum (map square (1 ... n))
  where
    square x = x*x
```

Feldspar

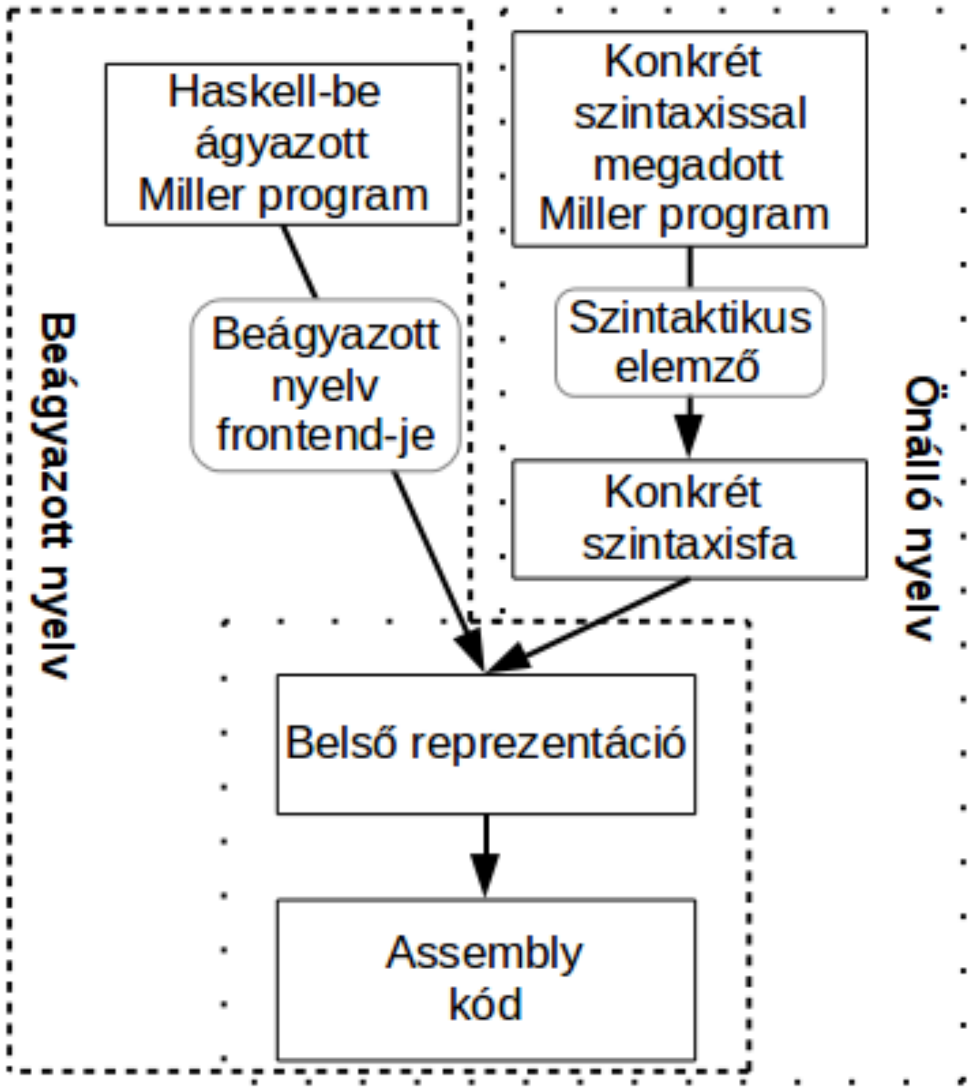

Feldspar forráskód
Frontend (magas szintű optimizációk)
Belső gráf reprezentáció
Gráf transzformáció (backend 1. fázis)
Absztrakt Imperatív kód
Optimalizációs plugin-ek (backend 2. fázis)
„Pretty printer" (backend 3. fázis)
HW-független C kód

Önálló nyelv készítése

Miller végső verziója

- saját szintaxis
- felhasznált elemek a beágyazott nyelvből
  - belső reprezentáció
  - backend — optimalizálás
  - backend — kódgenerálás
  - a teljes fejlesztés (beágyazott + önálló nyelv) 58%-a

Miller


Haskell-be ágyazott Miller program
Konkrét szintaxissal megadott Miller program
Beágyazott nyelv frontend-je
Szintaktikus elemző
Konkrét szintaxisfa
Beágyazott nyelv
Önálló nyelv
Belső reprezentáció
Assembly kód

Miller

Konkrét fejlesztés tapasztalatai

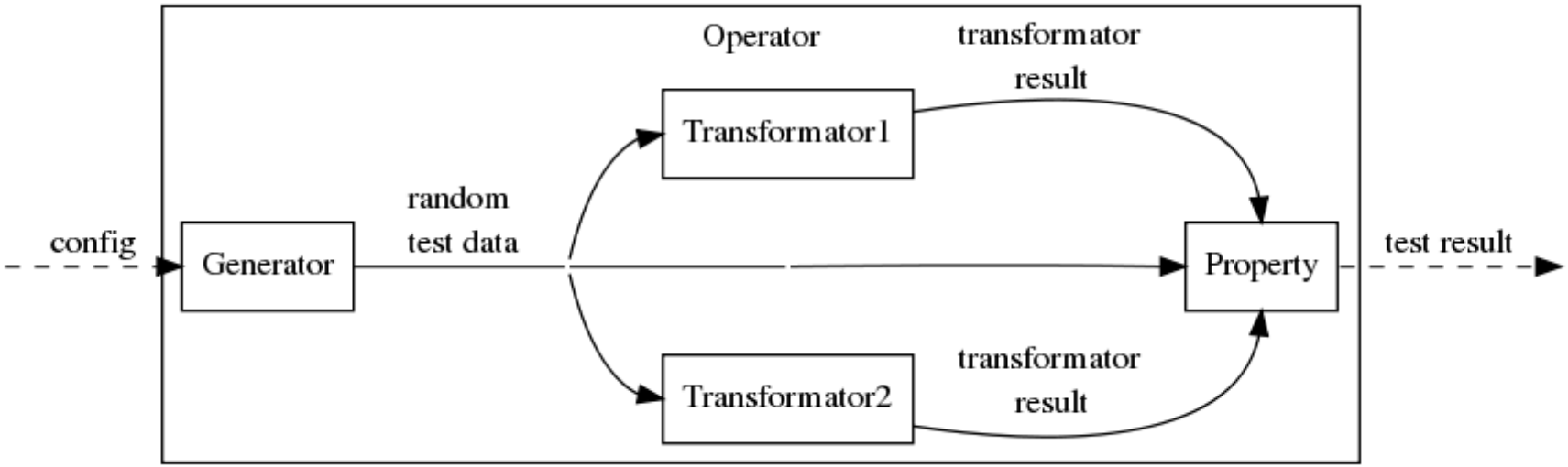
- kezdetben nyelvi elemek + belső reprezentáció
- konkrét szintaxis később
- gyors első verzió
- beágyazott változatnál csak funkcionális követelmények
- konkrét szintaxisfa $\Rightarrow$ absztrakt szintaxisfa
  - jelentős átalakítás
  - időigényes megvalósítás

Általános tesztelő keretrendszer

Motiváció

- könnyen kiterjeszthető új nyelvhez
- moduláris
- minimális költségű új tesztesetek
- integrálhatóság meglévő automatikus tesztelő eszközökhöz

A keretrendszer modellje



- *Generátor*: pl. `QuickCheck` generátor
- *Transzformátor*: a vizsgált eljárás (pl. fordító, interpreter) absztrakciója
- *Tulajdonság ellenőrző*: a kívánt tulajdonság vizsgálata a tesztadat és a transzformátorok eredményei alapján
- *Operátor*: teszt vezérlő

Generátor

- `QuickCheck` alapján
- fix hosszú heterogén lista
- végrehajtás
  - az operátor végrehajtja a generátorokat
  - a generált tesztadatot továbbítja a transzformátornak

Transzformátor

- függvény a tesztadatból egy kimeneti adatot generál
- nyelvfüggetlen transzformátor kombinátorok
  - `(»>)`
  - `pairT`
  - `listT`
  - ...

Tulajdonság ellenőrző

- esetfüggő
- előre definiált alapesetek
  - `strictEquality`
  - `elemWise`
  - `sumWise`
  - ...

Egy konkrét alkalmazás

Feldspar tesztelő

- Keretrendszer megvalósítása Haskellben
- A nyelv, az interpreter és a fordító tesztelése
  - Feldspar interpreter ?= Haskell interpreter (primitív operátorok, függvények)
  - Feldspar interpreter ?= Feldspar fordító
  - `FixedPoint` könyvtár pontosságának ellenőrzése
  - Feldspar-ban megírt algoritmusok ?= referencia C algoritmusok

Teszt példa

- Feldspar interpreter $\Leftrightarrow$ fordító
- viselkedés adott függvényekre

```
evalTransformator ::  testFun -> input -> result
```

- Feldspar interpreter végrehajtása adott függvényre és inputra

```
compilerTransformator ::  testFun -> input -> result
```

- Feldspar függvényből egyszerű C függvény lesz
- Szükség van egy generált `main()` függvényre
  - input beolvasás (stdin-ről)
  - létrehozott C függvény meghívása
  - eredmény kiírása (stdout-ra)
- C fájlok fordítása, eredmény külön folyamatként elindítása

Példa

Egy Feldspar függvény

```
overdrive :: DVector Float -> Data Float -> Data Float -> DVector Float
overdrive x mul bound = map (\x -> x * mul) $ map mapFn x
  where
    mapFn elem = (?) (elem > bound) (bound, elseBranch)
      where
        elseBranch = (?) (elem < - bound) (-bound,elem)
```

Egy teszteset

```
TestCase { tc_name = "overdrive (comp-eval)",
           gen = (vectorOf 16 genFloat, (genFloat, (genFloat, ()))),
           trans1 = compilerTransformator overdrive,
           trans2 = evalTransformator overdrive,
           prop = elemWise $ strictEquality }
```

RefactorErl

- Erlang statikus elemző és refaktoráló
  - 24 megvalósított transzformáció
    - kevesebb hibalehetőség
    - gyors
  - kódmegértés elősegítése
    - hívási gráf megjelenítése
    - függőség elemzés

Belső gráfreprezentáció

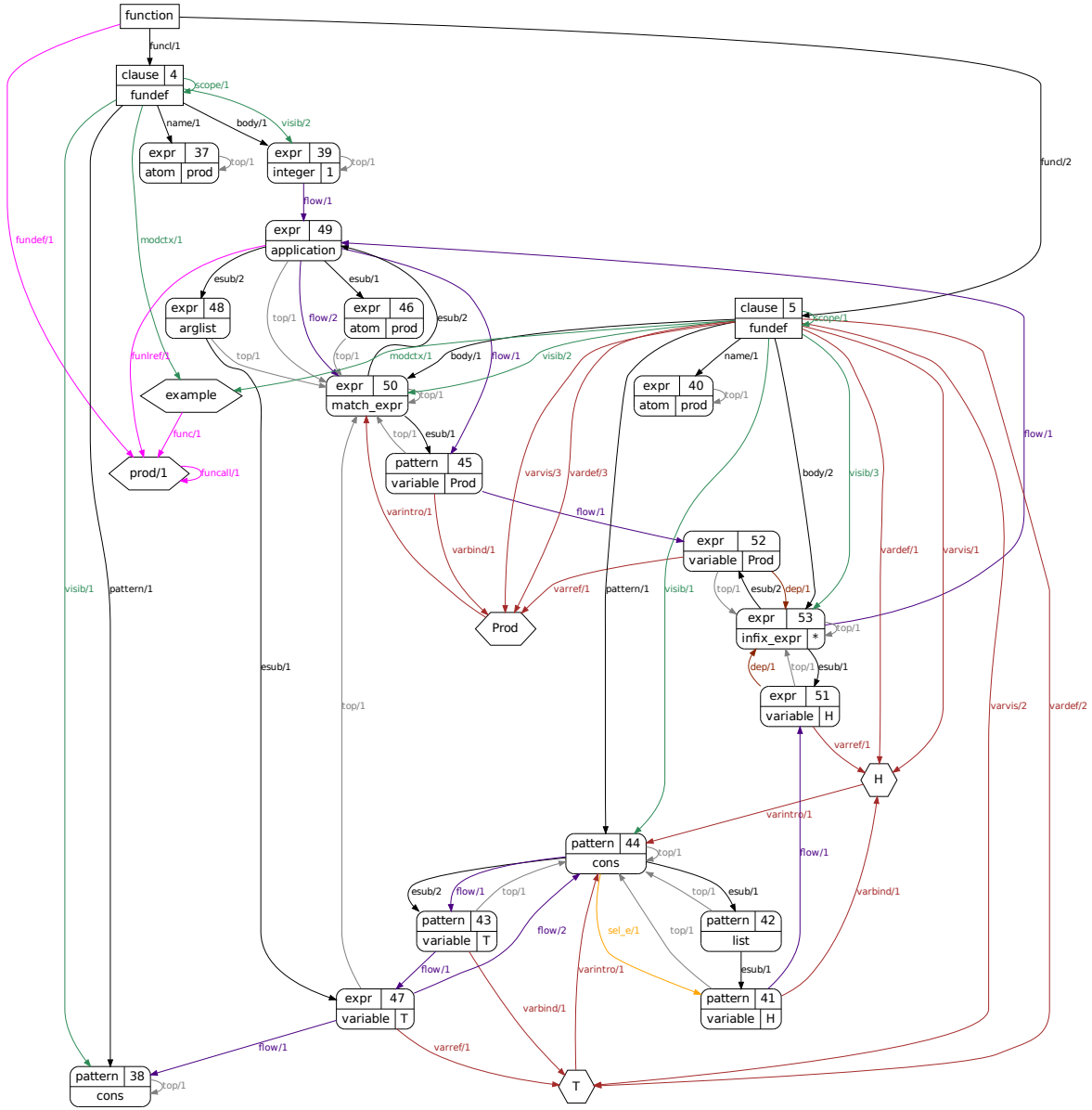
- Lexikális szint
  - tokenek
  - előfeldolgozás
  - kommenetek, whitespace-ek
- Szintaktikus szint
  - absztrakt szintaxisfa
  - fájlok
- Szemantikus szint
  - modulok, függvények, rekordok, változók csúcsai
  - függőségi információk

Példa program

```
-module(example).

prod([]) ->
    1;
prod([H|T]) ->
    Prod = prod(T),
    H * Prod.
```

A prod/1 függvény gráf reprezentációja



Kódminőség növelése

Regressziós teszt

- egyszerű elemző modulok
- gráf lekérdezések
- transzformációs lépések

Specifikáció alapján végzett tesztelés

- komplex szemantikus elemzés
- transzformációs lépések

Specifikáció alapján végzett tesztelés

A belső gráfreprezentáció felhasználása

- formális specifikáció $\Longrightarrow$ gráfreprezentáció tulajdonságai
- konzisztencia ellenőrzés
- véletlenszerűen generált Erlang modulok tesztadatként

Mintaillesztő kifejezés adatfolyam szabályai

Erlang kifejezés:

e_0:

$$p = e$$

Adatfolyam szabályok:

$$e \overset{flow}{\rightarrow} e_0$$

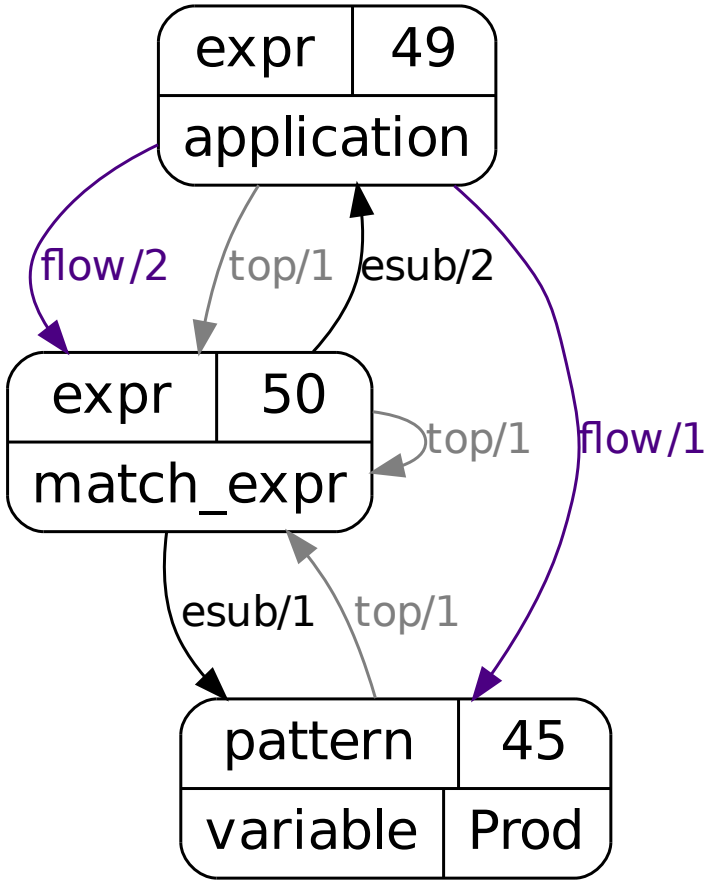
$$e \overset{flow}{\rightarrow} p$$

Példaprogram

```
-module(example).

prod([]) ->
    1;
prod([H|T]) ->
     Prod = prod(T),
    H * Prod.
```

A prod/1 függvény reprezentációs gráfjának részlete



```
props(MatchExpr) ->
    case checkTwoChild(MatchExpr) of
       true -> Ch1 = getFirstCh(MatchExpr),
               Ch2 = getSecondCh(MatchExpr),
               check_matchExpr(MatchExpr, Ch1, Ch2);
       false -> {error_in_matchexpr, MatchExpr}
    end.
```

```
check_matchExpr(MatchExpr, Ch1, Ch2) ->
    Ch2Flows = getFlows(Ch2),
    case lists:member(Ch1, Ch2Flows) and
           lists:member(MatchExpr, Ch2Flows) of
        true  -> ok;
        false -> {error_in_matchexpr, MatchExpr}
    end.
```

Program generálás

Véletlenszerűen generált Erlang programok

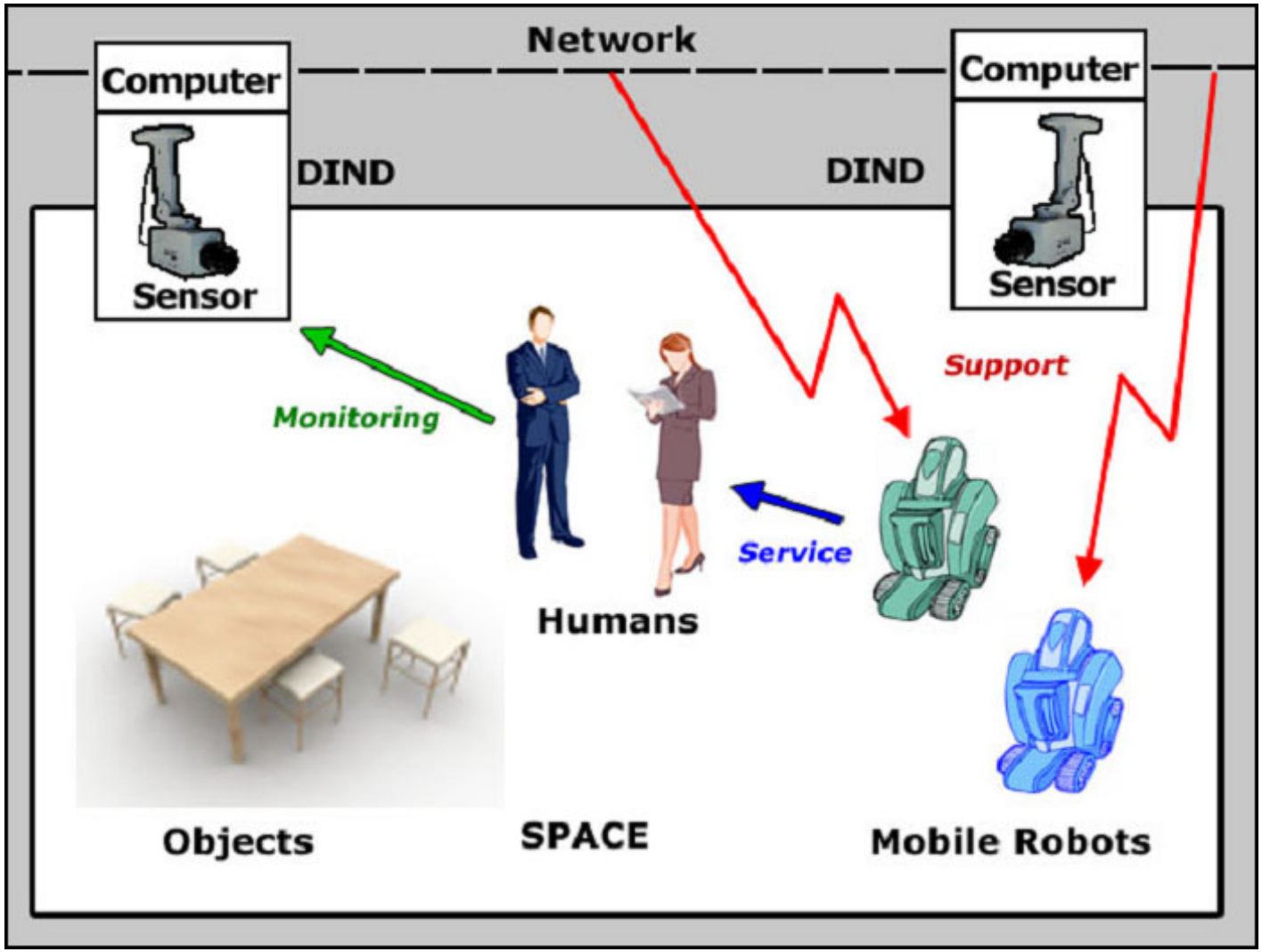
- formális attribútum grammatika

Adott tulajdonsághoz alakítható

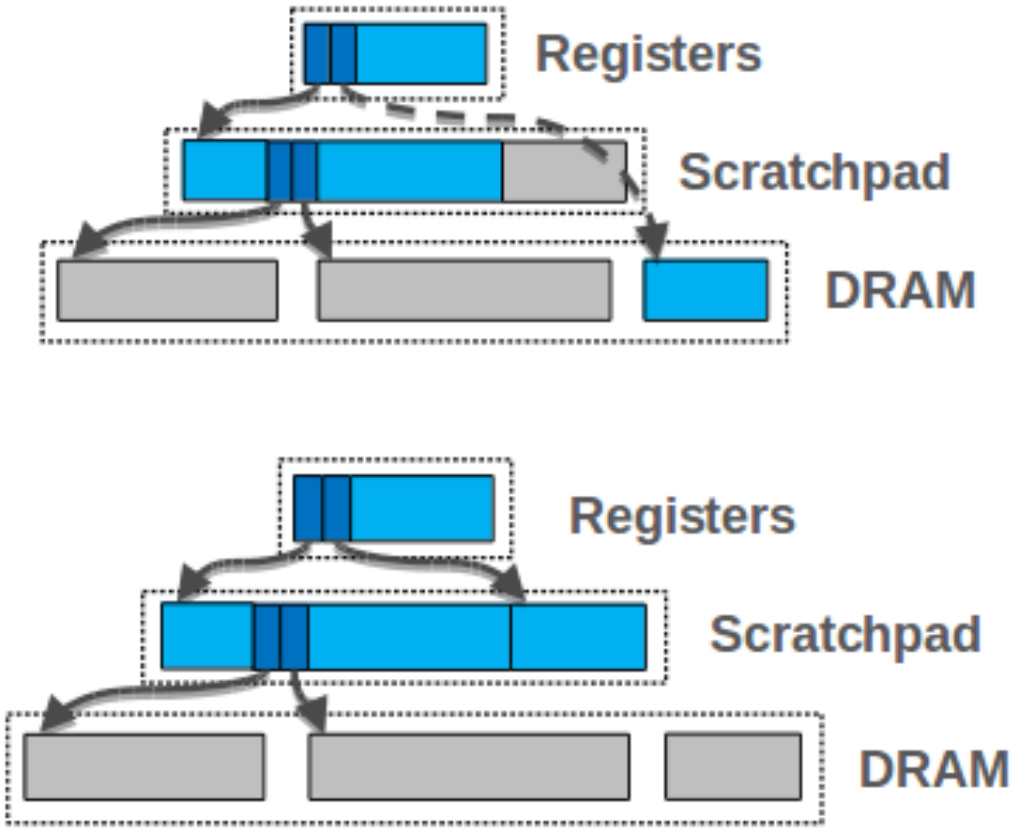
- fordítható programok
- függvények (rekúrzió nélkül)
- egymástól függő modulok
- ...

Véletlenszerű értékek a QuickCheck alapján

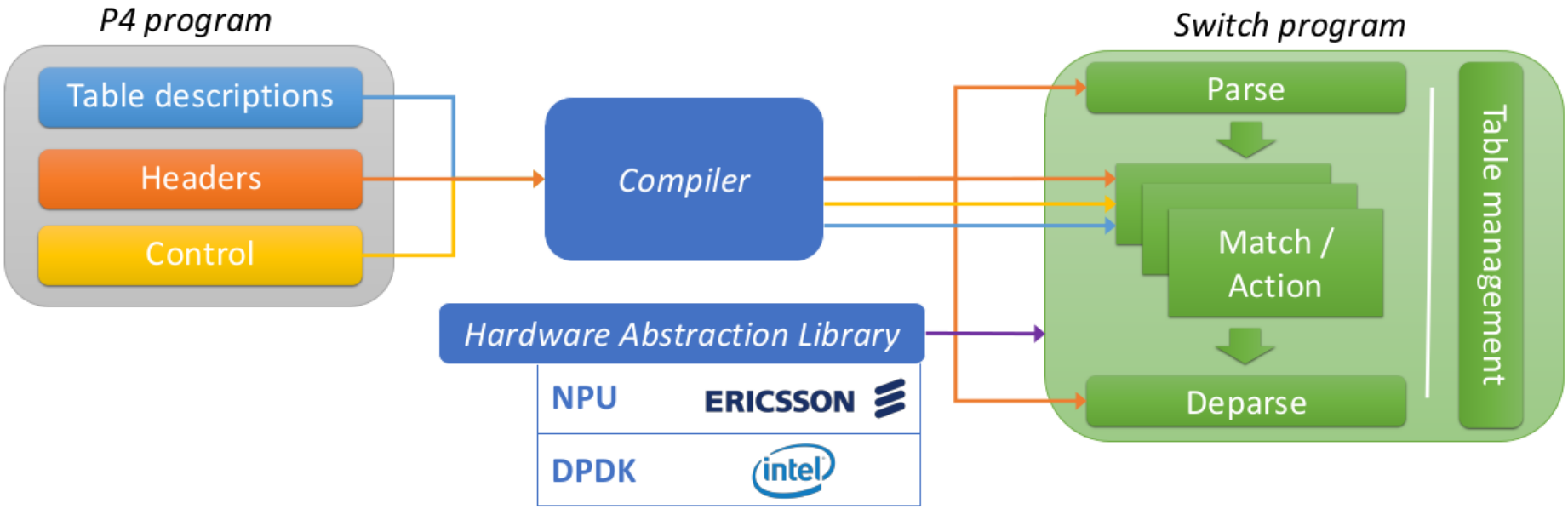
Ellenőrzött mobil kód ISpace környezetben



Konfigurálható adatszerkezetek



P4 fordító


P4 program
Table descriptions
Headers
Control
Compiler
Hardware Abstraction Library
NPU
ERICSSON
DPDK
intel
Switch program
Parse
Match / Action
Deparse
Table management

Összegzés

Forráskód feldolgozás

- Nyelvek kibővítése, új nyelvek készítése, refaktoráló eszközök
- Elemzés, optimalizálás, refaktorálás, fordítás

Tesztelés

- Tesztelő keretrendszer, Feldspar tesztelő
- Refaktorerl tesztelő

Adatok

Publikációk (PhD fokozat megszerzése óta)

- 16 kapcsolódó publikáció
- 3 további publikáció
- 1 elfogadott, megjelenés alatt álló publikáció
- 3 beküldött, bírálat alatt álló publikáció

Oktatás

- gyakorlatvezetés 1998-tól
- előadás tartás 2004-től
- szoft. tech. labor 2009-től
- PhD
  - témavezető: 3 hallgató
  - társtémavezető: 2 hallgató