Xoc, an Extension-Oriented Compiler for Systems Programming

R.Cox, T.Bergan, A.Clements, F.Kaashoek, E.Kohler. 2008
http://pdos.csail.mit.edu/xoc/asplos08.pdf

Laiendustele orienteeritud kompilaator süsteemide programmeerimiseks

Jaak Ristioja
2009 Tartu

Ettekande kava

- Sissejuhatus
- **Xoc**'i liides keele laiendusteks
- **Xoc**'i implementatsioon
- **Xoc**'i kasutusjuhtumid (*case studies*)
- Demo?

Sissejuhatus

- Lisandused programmeerimiskeeltele
  - Sündmuspõhine programmeerimine (*event-driven programming*)
    - Tame (Krohn jt 2007), Tamer (Kohler 2007)
  - Hajussüsteemide programmeerimine
    - Mace (Killian jt 2007)
  - Koodianalüüs
    - Sparse (Torvalds, Triplett 2007), Linux'i kerneli C *front-end*

Sissejuhatus

- Tüüpiline realisatsioon:
  - *Front-end*
    - Teisaldab laiendustega keele lähtekoodi standardsete keelekonstruktsioonidega lähtekoodiks ning siis kompileerib/interpreteerib
- Probleemid:
  - Üksikud laiendused on keerulised implementeerida, võtavad palju kõvaketta ruumi, mälu ja aega
  - Raske kasutada korraga mitmeid laiendusi

Sissejuhatus

Värskemaid ideid:

- Necula jt 2002; Nystrom jt 2003; Grimm 2006; Visser 2004:
  - Laiendatav kompilaator, mida saab järk-järgult laiendada
  - Probleemid:
    - Laiendusi raske komponeerida (üks laiendatud kompilaatori ei pruugi teise sarnase sisendit vastu võtta)
- Van Wyk jt 2007; Nystrom jt 2006
  - Laienduste kirjutajad peavad iga üksiku kompositsiooni kirjeldama

Sissejuhatus

- **Xoc**
  - Laiendusi laetakse dünaamiliselt
  - Alustatakse baaskeele kompilaatorist
    - Laienduste lisamisel ei genereerita täiesti uut kompilaatorit

Xoc'i liides keele laiendusteks

- **Xoc** tõlgib lähtekoodi lähtekoodiks:
  - Loeb C-keele lähtekoodi, kus võib olla kasutatud C-keele laiendusi,
  - analüüsib seda,
  - kompileerib võrdväärseks C koodiks ning
  - annab kompileerimiseks ette GCC'le.
- Käsureal on võimalik määrata laiendusi, mida **Xoc** peab lähtefaili puhul kasutama.

Xoc'i liides keele laiendusteks

- **Xoc** defineerib C kontekstivaba baasgrammatika:

```
extensible grammar C99 {
    ...
    %left Add Shift
    %precedence Add > Shift
    ...
    expr: name
        | expr “+” expr    [Add]
        | expr “-” expr    [Add]
        | expr “>>” expr   [Shift]
        | expr “<<” expr   [Shift]
    ...
}
```

Xoc'i liides keele laiendusteks

- Laiendused täiendavad baasgrammatikat, lisades sellele uusi reegleid.
- Näide bitiviisilise pööramise (*bit-wise rotate*) operaatoritest:

```
grammar Xrotate extends C99 {

    expr: expr “<<<” expr   [Shift]

        | expr “>>>” expr   [Shift];

}
```

Xoc'i liides keele laiendusteks

- **Xoc** võimaldab luua uusi prioriteeditasemeid luua olemasolevate vahele
- Näide (keele Alef iteratsioonioperaator “**: :**”):

```
grammar XAlefIter extends C99 {
    %right AlefIter;
    %priority Shift > AlefIter > Relational;
    expr: expr “::” expr   [AlefIter];
}
```

- Näide iteratsioonioperaatori kasutamisest keeles Alef:

```
void copy(byte *to, byte *from) {
    to[0::strlen(from)-1] = *from++;
}
```

Xoc'i liides keele laiendusteks

- Eelnevaid definitsioone kasutades suudab **Xoc** vahet teha neljal võimalikul grammatikal:
  - C99
  - C99 + XRotate
  - C99 + XAlefIter
  - C99 + XRotate + XAlefIter
    - see on sama mis C99 + XAlefIter + XRotate
- Parsimisel kasutab **Xoc** nii C99 kui ka kõiki laetud laiendusi.

Xoc'i liides keele laiendusteks

- **Xoc**'i tüübisüsteem suudab väljendada, millisele grammatikale mingi abstraktse süntaksi puu osa vastab.

  See on kasulik defineerimaks, mis süntaksit mõni funktsioon ootab.

  - Näiteks funktsioon, mis analüüsib ainult standardse C99 avaldisi võiks nõuda sisendina avaldist tüübiga C99.expr.
  - Funktsioon, mis suudab analüüsida ka bitiviisiliste pööretega avaldisi võiks sisendina aksepteerida (C99+XRotate).expr tüüpi avaldisi.
- Samuti eksisteerib jokkergrammatika (*wildcard*) tähistamaks kõiki staatiliselt veel teadmata grammatikaid
  - C99+XRotate+?

Xoc'i liides keele laiendusteks

- **Xoc** kasutab kontekstivabu grammatikaid.
- Seetõttu pole alati võimalik leida ühest abstraktse süntaksi puud.
- Hetkel tuvastab **Xoc** mitmesust transleerimisel, st alles siis kui programmeerija on kasutanud vastavat keelekonstruktsiooni.
- **Xoc** leiab kõik võimalikud süntaksipuud ja tagastab vea.

Xoc'i liides keele laiendusteks

- Peale parsimist käivad **Xoc** ja tema laiendused üle süntaksipuu, arvutavad välja selle omadused ning võivad seda teisendada.
- Kasutamata traditsionaalseid andmestruktuure, kasutavad **Xoc**'i laiendused süntaksipuule viitamiseks konkreetset süntaksit (programmeerimiskeele süntaksit).

Xoc'i liides keele laiendusteks

- Süntaksi-mustrid (*syntax patterns*)
- *Destruktureerimise* (*destructuring*) avaldis:
  - Näiteks avaldis `ast ~ expr{\a <<< \b}` viitab laienduse XRotate esimesele reeglile.
    - Väärtustub kas tõeseks või vääraks
    - Muutujad `a` ja `b` seostatakse vastavate `ast` alampuudega.

Xoc'i liides keele laiendusteks

- Süntaksi-mustrid (*syntax patterns*)
- *Restruktureerimise* (*restructuring*) avaldis:
  - Näiteks avaldis `'expr{\a <<< \b}` viitab samuti laienduse XRotate esimesele reeglile.
    - Tagastab uue süntaksipuu, mille juureks on vastav avaldis.
    - Tagastatava süntaksipuu alampuudeks on muutujatega a ja b seotud alampuud.

Xoc'i liides keele laiendusteks

- Näide. Mitmekordse rotatsiooni ümberkirjutamine üheks rotatsiooniks:

```
if (ast ~ expr{\a <<<< \b <<< \c})
    ast = 'expr{\a <<< (\b + \c)};
```

Xoc'i liides keele laiendusteks

- Eeldasime, et `ast` on vähemalt C99+XRotate, mistõttu saime kasutada süntaksi mustrites <<< operaatorit.
- Samuti tähendas see, et `\a` ja `\b` olid vähemalt C99+XRotate.
- Kui grammatikaks oleks C99+XRotate+?, siis võiks `\a` ja `\b` olla suvaliste laiendusega süntaksipuud.

Xoc'i liides keele laiendusteks

- Näited:

```
C99.expr{\a <<< \b}
```

  - vigane, sest C99 ei sisalda <<< operaatorit
- Järgnevad read pole võrdsed, sest esimene neist ei lubaks ülejäänud laienduste konstruktsioone `\a` ega `\b` sees.

```
(C99+XRotate).expr{\a <<< \b}
(C99+XRotate+?).expr{\a <<< \b}
```

Xoc'i liides keele laiendusteks

- Mugandused **Xoc**'is:
  - C = C99+?
  - Staatiline tüübiinformatsioon võimaldab kirjutada

    ```
    ast ~ expr{\a <<< \b}
    ```

    asemel

    ```
    ast ~ {\a <<< \b}
    ```

    kui on teada, et `ast` on deklareeritud kui `C.expr`.
  - Võimalik lihtsasti konverteerida muutujaid süntaksipuu tippudeks (täisarvud, sõned jne).
  - Ja muu, *“special processing of syntax patterns”*

Xoc'i liides keele laiendusteks

- Paljud kompilaatorid analüüsivad programmi astmeliselt, näiteks:
  - Esiteks muutujate skoobianalüüs
  - Teiseks tüübikontroll
  - Kolmandaks konstantide väärtustamine
  - jne
- Laiendused võivad vajada mõne taolise analüüsi tulemusi, või isegi läbi viia mingit oma analüüsi.
- Teistes laiendatud kompilaatorites on see astmelisus ilmutatud ning laiendused peavad deklareerima, kuidas nad sellesse struktuuri mahuvad.

Xoc'i liides keele laiendusteks

- **Xoc**'is taoline astmelisus puudub
- Kasutatakse laiska väärtustamist süntaksipuu tippude atribuutide jaoks, nt .type.
  - Esimesel kasutamisel arvutatakse välja
  - Järgmistel kordadel võetakse väärtus juba puhvrist, sest süntaksipuid ei saa muuta (*immutable*).
- Laisa väärtustamise tõttu ei ole oluline, mis järjekorras tulemused arvutatakse.
- Iga atribuudi defineerib tavaline **Xoc**'i kood, mistõttu neid võib laiendada sarnaselt grammatikale.

Xoc'i liides keele laiendusteks

```
attribute
type(term: ptr C.expr): ptr Type
{
  switch(term){
    case ~{\a << \b} || ~{\a >> \b}:
      if (a.type.isinteger && b.type.isinteger)
        return promoteunary(a.type);
      error(term.line, “non-integer shift”);
      return nil;
    // … other cases …
  }
  error(term.line, “cannot type check expression”);
  return nil;
}
```

Xoc'i liides keele laiendusteks

```
extend attribute
type (term: ptr.C.expr): ptr Type
{
  switch(term){
    case ~{\a <<< \b} || ~{\a >>> \b}:
      if(a.type.isinteger && b.type.isinteger)
        return promoteunary(a.type);
      error(term.line, “non-integer rotate”);
      return nil;
  }
  return default(term);
}
```

Xoc'i liides keele laiendusteks

- Kui mitu laiendust laiendab mingit atribuuti, ühendatakse laiendused ahelasse kasutades `default` väljakutseid.
- Sedasi arvutatakse atribuudid ülevalt alla, kuid on võimaldatud ka muud lähenemised.
  - Süntaksipuu tippudel on defineeritud sisseehitatud atribuudid `parent`, `prev` ja `next`, mis on siis vastavalt tipu vanem, vasak ja parem naaber.
    - Näiteks muutujate skoobi leidmise tarvis.
    - Ka need atribuudid on laisad.

Xoc'i liides keele laiendusteks

- Pärast esialgse sisendi parsimist vaatab **Xoc** süntaksipuu juure atribuudi `welltyped` järgi, kas terve puu on tüübivigadest puhas.
- Laiendused, mis lisavad keelde uusi lauseid laiendavad ka `welltyped` atribuuti.
- Avaldistel on eelpoolmainitud `type` atribuut.
- Tüübikontroll sõltub ka skoobis olevatest muutujatest, seega on iga süntaksipuu tipuga seotud
  - atribuut `vars` – skoobis muutujad tippu sisenemisel
  - atribuut `vars_out` – skoobis muutujat tipust väljumisel
- Kui kogu süntaksipuu juure `welltyped` atribuut on tõene, kasutab **Xoc** globaalsete muutujatena muutujaid juure atribuudis `vars_out`.
- Iga globaalse muutuja `compiled` atribuut sisaldab sellele ekvivalentset C koodi, mis moodustavadki väljastatava C lähtekoodi.

Xoc'i liides keele laiendusteks

- Muutujate hõivamine (*capture*)
  - Kui mingi viide (*reference*) muutujale `x` kopeeritakse uude süntaksipuusse, peaks see vaikimisi ikkagi samale muutujale viitama, isegi kui uue asukoha skoobis juba on samanimeline muutuja.
  - **Xoc** väldib taolist olukorda ehk hoiab “hügieeni” (*hygiene* – Kohlbecker jt 1986)
    - Uutel süntaksipuu tippudel on `copiedfrom` attribuut, mis viitab tipule, millest see kopeeriti.
    - Muutuja uus nimi võidakse saada `copiedfrom.sym` atribuudist.

      (siis kui `copiedfrom` ei ole null)
    - **Xoc**'i C väljundinga tegelev osa nimetab muutujad vastavalt ümber.

Xoc'i liides keele laiendusteks

- Muud liidesed:
  - Laiendatavad andmestruktuurid
    - Laiendused võivad **Xoc**'i sisemistele andmestruktuuridele välju lisada.
    - Erinevate laienduste lisatud väljad on vastavalt skoobitud nii, et laiendused teineteise väljadele ligi ei pääse (ei ole konflikte ka samade nimede puhul)
  - Laiendatavad funktsioonid
    - **Xoc**'i funktsioone on võimalik laiendada samamoodi nagu atribuute.

Xoc'i liides keele laiendusteks

```
extend fn
xcanconvert(from: ptr Type, to: ptr Type): bool
{
  if((from ~ {unsigned char*} && to ~ {char*})
  || (from ~ {char*} && to ~ {unsigned char*}))
    return true;
  return default(from, to);
}
```

Xoc'i liides keele laiendusteks

- Muud liidesed:
  - Üldiseks süntaksipuu läbimiseks ja ümberkirjutamiseks on olemas funktsioonid
    - `astsplit` – tagastab süntaksipuu tipu otsesed alamtipud massiivina
    - `astjoin` – tagastab uue süntaksipuu etteantud juure ja alamtippudega.
  - Kontrollvoo-analüüsi moodul (*control flow analysis module*)

Xoc'i implementatsioon

- Kolmas **Xoc**'i prototüüp kirjutatud keeles **Zeta**.
  - Eelnevad prototüübid olid kirjutatud laiendatud C keeles.
  - Zetas olevat palju lihtsam uusi asju testida – **Xoc**'ile külge pookida ja eemaldada.
- Jookseb interpretaatoris “zeta”.
  - Interpretaator on aeglane
  - Ei kompileerunud 64-bitiseks.

Xoc'i implementatsioon

- GLR parser (*Generalized Left-to-right Rightmost derivation parser*)
  - Suudab toime tulla mitte-determineeritud mitmeste grammatikatega.
    - Tagastab kõik võimalikud väljundid.
    - On võimatu kirjutada kontekstivaba grammatikat, millega GLR parser ei suudaks toime tulla.
    - Teiste LR parserite korral võivad koos kasutatud keelelaiendused konflikte tekitada.
- C's defineeritavate muutujate ja tüüpide nimede eristamisel on **Xoc**'is vastav funktsioon
  - Kui eristamine siiski ei õnnestu tagastatakse viga.

Xoc'i implementatsioon

- Abstraktse süntaksipuu tipud
  - `AstRule`
    - sisaldab viitasid vastavale grammatika reeglile ja alamtippudele
  - `AstString`
    - lekseem (näiteks “`while`” või “==”)
  - `AstSlot`
    - tühi pesa nagu `\a` süntaksi mustris.
- Süntaksipuu tippude tüübid (nagu näiteks C99+XRotate) on võimalik täpselt välja arvutada süntaksipuu ehitamisel.

Xoc'i implementatsioon

- Süntaksimustrid (*syntax patterns*)
  - GLR parseringa esialgset koodi parsides, luuakse mallid destruktureerimiseks ja restruktureerimiseks
  - Igasse pesasse võivad sattuda erinevad väärtused
    - Xoc valib madalaima GLR parseri leitud puu.
    - On võimalik pesade tüüpi eraldi annoteerida:

      ```
      \a::number
      ```
    - Madalaim puu on tavaliselt alati see, mida tahetakse.

Xoc'i kasutusjuhtumid

- Sparse, Linuxi kerneli kontrollija - xsparse

  kontrollib koodi stiili ja laiendab C tüübisüsteemi:

  - Lisab viitadele ehk pointeritele aadressiruumi omaduse (*user-space* või *kernel-space*)
    - Keelab ligipääsu kasutajaruumi (*user-space*) viitavate viitade poolt viidatavatele väärtustele (*dereference*).
    - Hoiatab erinevate aadressiruumide viitade *cast*imisel.
  - Lisab funktsioonidele konteksti, kontrollimaks kompileerimisel resursside lukustamist ja vabastamist.
    - `welltyped` atribuudite abil kontrollib konteksti
    - `compiled` atribuudite abil antakse ka lõplik C-kood edasi vookontrollile, et analüüsida ka teiste võimalike laienduste tekitatud koodi.

Xoc'i kasutusjuhtumid

xsparse näide tüübikontrollist:

```
extend attribute
type(term: ptr C.expr): ptr Type
{
  t := default(term);
  // kontroll
  return t;
}
```

Xoc'i kasutusjuhtumid

- Sparse
  - 25 000 rida koodi
  - do_mounts.c (600 kB) – ~0,1s
- xsparse
  - 345 rida koodi (toetub suuresti **Xoc**'ile)
  - do_mounts.c (600 kB) – ~15s (Zeta interpretaator)
  - Töötab hästi ka teiste keelelaiendustega

Xoc'i kasutusjuhtumid

- xlambda

```
void alphabetize(int ignorecase, char **str, int nstr)
{
  qsort(str, nstr, sizeof(char*),
    fn int cmp(const void *va, const void *vb) {
      const char **a = va, **b = vb;
      if (ignorecase)
        return strcasecmp(*a, *b);
      return strcmp(*a, *b);
  });
}
```

Xoc'i kasutusjuhtumid

- xlambda genereeritud kood:

```
int lambda_cmp(struct env_cmp *env,
  const void *a, …)
{
  if (env->ignorecase)
  …
```

- xlambda laiendus:
  - 170 rida Zeta koodi

Xoc'i kasutusjuhtumid

```
void foreach(char **str, int nstr, void(*f)(char*)) {
  f(str[0::nstr]);
}

int main(int argc, char **argv) {
  while (getline()) {
    foreach(argv+1, argc-1,
      fn void check(char *pat) {
        if (it =~ pat) printf(“%s\n”, $0.str);
      });
  }
}
```

Xoc'i kasutusjuhtumid

Name	Lines	Description
`xaif`	50	Make `if` and `while` anaphoric, as in *On Lisp* (Graham 1996).
`xalef-check`	24	Add `check` statement as in Alef (Winterbottom 1995).
`xalef-iter`	196	Add iterator expressions as in Alef.
`xgnu-asm`	47	Parse (but do not analyze) GNU inline assembly.
`xgnu-caserange`	61	Allow ranges in `case` labels (`case 0 ... 9:`).
`xgnu-conditional`	14	GNU binary conditional operator `?:`.
`xgnu-minmax`	24	GNU min and max operators <?, >?, <?=, and >?=.
`xgnu-typeof`	33	GNU `typeof` type specifier (`typeof(q) p = q;`).
`xlambda`	170	Heap-allocated lexical closures that are compatible with regular function pointers.
`xloop`	168	Labeled `break` and `continue`, as in Java and Perl.
`xpcre`	452	Perl-like syntax for the PCRE regular expression library, including flow-sensitive checks for out-of-range submatch references.
`xrotate`	34	Rotate operators <<< and >>>.
`xsparse`	345	Workalike for the Sparse program checker (Torvalds and Triplett 2007). (80 lines for type checking, 245 lines for flow checking.)
`xtame`	516	Tame style event-driven programming (Krohn et al. 2007).
`xvault`	641	An implementation of Vault's flow-sensitive type system (DeLine and Fahndrich 2001) for C.

Figure 3. Extensions written using xoc. The lines column counts non-comment source lines.

Otse artiklist: http://pdos.csail.mit.edu/xoc/asplos08.pdf

Xoc'i kasutusjuhtumid

```
XRotate

extend attribute
compiled(term: ptr C.expr): ptr COutput.expr
{
  switch(term){
    case ~{\a <<< \b}:
      n := a.type.sizeof * 8;
      return ‘C.expr{
               ({ \(a.type) x = \a;
                  \(b.type) y = \b;
                  (x<<y) | (x>>(\n-y)); })
             }.compiled;
    case ~{\a >>> \b}:
      n := a.type.sizeof * 8;
      return ‘C.expr{\a <<< (\n-\b)}.compiled;
  }
  return default(term);
}
```

Üks niisama mõttetu slaid

Demo?

Xoc ja sellega seonduv on saadaval aadressil: http://pdos.csail.mit.edu/xoc/

EOF