

# Effect Types and Region-Based Memory Management

Søren Vrist

DIKU

10. Januar 2006

Bygger krakilsk på kapitel 3 fra ATAPL



- 1 Valueflow vha label-typning
  - Tagget sprog
  - Flowinformation vha. labels
  - Korrekthed
  - Inferens
- 2 Effekter
  - Scoped tagget sprog (STL)
  - Scoped effect typet sprog (ETL)
- 3 Regionsbaseret lagerhåndtering
  - Region-Annotated sprog (RAL)
  - Annotering af programmer
- 4 Tofte-Talpin typesystemet
  - Sundhed

# Agenda

- 1 Valueflow vha label-typning
  - Tagget sprog
  - Flowinformation vha. labels
  - Korrekthed
  - Inferens
- 2 Effekter
  - Scoped tagget sprog (STL)
  - Scoped effect typet sprog (ETL)
- 3 Regionsbaseret lagerhåndtering
  - Region-Annotated sprog (RAL)
  - Annotering af programmer
- 4 Tofte-Talpin typesystemet
  - Sundhed



# Indledning

Sproget BL er grundlæggende for de de følgende udvidelse mht til tagging, regions, effekter osv.

- Simpelt typet lambda-kalkyle
- Generel recursion
- Bool
- Call-by-value semantik



## Tagget sprog

Udvider BL med eksplicit tagging og untagging til sproget TL.

- Label-variablene  $\rho$  er et tælbart set af label-variable  $\rho_1, \rho_2 \dots$
- `t at p` tagger værdien af en term  $t$  med en label  $p$  og giver en værdi  $\langle v \rangle_\rho$
- `t ! p` evaluerer  $t$  til en tagget værdi  $\langle v \rangle_\rho$  returnere  $v$  hvis  $p$  er det rette label.
- label mismatch giver en stuck-state, evalueringen "slår fejl".
- Type systemet i TL garanterer at evalueringer aldrig slår fejl.



## Tagget sprog - 2

Bemærk:

- Flere subtermer af en term kan have samme label.
- En label må kun optræde engang i et program, alligevel kan den godt tage flere værdier runtime.
- Labeler lader os se forskel på værdier fra forskellige steder selvom værdierne er "ens".
- Der er forskel på  $v \text{ at } p$  og  $\langle v \rangle_p$



## Flowinformation vha. labels

Definerer erasure og completion

- Erasure  $\|\cdot\|$ , dvs. når  $t \in TL$  er  $\|t\|$  erasure af  $t$ .  
Bemærk at  $\|t\|$  er en BL-term.
- En TL-term  $t'$  er completion af en BL-term  $t$  når  $\|t'\| = t$
- Erasure er lukket under erstatning.(Substitution)



## Flowinformation vha. labels - 2

Con/decon completion:

- Værdier tagges når de laves
- Tags kontrolleres og fjernes lige før den underliggende værdi fjernes.  
le: I if-then-else betingelsen og funktionen i en funktions anvendelse.
- Derved fortæller  $p$  i  $t ! p$  hvilke værdi udtryk der kan have skabt værdien af  $t$ .  
Under forudsætning af at ingen termer ender i stuck-state under evaluering.



## Korrekthed

Man kan se på en TL-term som en udgave af BL med ekstra muligheder, intuitivt fordi TL udfører de samme udregninger som BL med nogle få indskudte ekstreme trin til label-reduktion. Dette vises med et korrekthedsbevis vha:

- Simulation
- Betinget korrekthed
- Preservation
- Progress
- Sundhed



## Korrekthed - 2

Korrekthed:

- 1  $t \uparrow$  hviss  $\|t\| \uparrow$
- 2  $\|t\| \rightarrow^{BL*} \|v\|$  hviss der eksistere en TL-værdi  $v'$  med  $\|v'\| = \|v\|$  og  $t \rightarrow^{T*} v'$



# Inferens

- Ønsker at finde en principal løsning.  
le.: flest muligt forskellige labels
- En con/decon skabelon for en BL-term er en con/decon completion af t hvor alle labels kun optræder een gang.
- Alle con/decon completions der er TL typebar er en instans af en sådan template med erstatninger af labels til labels.
- Muligt at vise at en erstatning giver en veltypet con/decon completion hviss det opfylder ligheds-constraints.
- Det er muligt i linær tid at beregne disse ligheds-constraints og tilsvarende i linær tid er det muligt at finde en mest generel løsning. Analog til simpel type inferens.



# Agenda

- 1 Valueflow vha label-typning
  - Tagget sprog
  - Flowinformation vha. labels
  - Korrekthed
  - Inferens
- 2 Effekter
  - Scoped tagget sprog (STL)
  - Scoped effect typet sprog (ETL)
- 3 Regionsbaseret lagerhåndtering
  - Region-Annotated sprog (RAL)
  - Annotering af programmer
- 4 Tofte-Talpin typesystemet
  - Sundhed

## Scoped tagget sprog (STL)

Omformer labels/tagging til regioner og region adgange ved at se taggedede værdier som pointere ind i en region hvor værdien ligger

- Ser at hvis  $\Gamma$  og  $T$  ikke indeholder  $\rho$  men bruges i  $t$ , bruger kontekst før og efter ikke  $\rho$  og alt i  $\rho$  kan slettes efterfølgende
- Introducere "scopede regions" med `new  $\rho.t$`  som binder en regionvariable i  $t$  til en ny allokeret region og efter  $t$  er evalueret til en værdi deallokeres regionen igen.
- Adgange til en deallokeret værdi `< $v$ >.!•` er stuek.
- STL er ikke sund.



## Scoped effect typet sprog

$$\Gamma \vdash t :^{\varphi} T \quad (1)$$

$\varphi$  er effekten og  $^{\varphi}T$  er effekt og type, effekttype.  
 $t$  har den observerbare effekt  $\varphi$  og ender med typen  $T$ .

- Introducerer ETL.
- En effekt er et endelig sæt af af regionvariabler (uden •)
- Rækkefølgende af effekter i  $\varphi$  uden betydning
- I modsætning til STL er ETL sund.



# Agenda

- 1 Valueflow vha label-typning
  - Tagget sprog
  - Flowinformation vha. labels
  - Korrekthed
  - Inferens
- 2 Effekter
  - Scoped tagget sprog (STL)
  - Scoped effect typet sprog (ETL)
- 3 **Regionsbaseret lagerhåndtering**
  - **Region-Annotated sprog (RAL)**
  - **Annotering af programmer**
- 4 Tofte-Talpin typesystemet
  - Sundhed



# Indledning

- Historisk set er lagerhåndtering foretaget explicit af programmør eller automatisk med garbage collection.
- Introducere her en måde med explicit annotering af allokering og deallokering
- Typesystem sikrer deallokering
- Automatisk inferens kan være muligt ved compile-time
- En region er en sub-hob med hob-allokerede værdier
- Hoben er en hob af regioner



## Region-Annotated sprog (RAL)

- Lambda kalkyle
- Udvidet med fix, bool, og eksplicite regions annoteringer
- new udfører implicit alfa konverteringer for at undgå indfangning
- RE-DEALLOC virker på både allokering-udtryk og allokerede værdier. At prøve at allokere en værdi til en ikke eksisterende region giver først fejl ved eksekvering. Tilsvarende med region application.
- $eval_R$  evaluerer termer fra RAL til  $\{tt, ff, \perp, wrong\}$



## Region-Annotated sprog (RAL) - 2

Regionsabstraktioner:

- Svarer til abstraktioner som vi kender dem, bare over regioner i stedet for.
- Virker over værdier eller "næsten-værdier".
- Region polymorfi: Regionsabstraktioner tillader at man parameteriserer en funktion over de nødvendige regioner.
- Bruger region polymorphic recursion



## Annotering af programmer

- Analogt til TL vs. BL er det muligt at gå fra en RAL-term til BL-term ved at fjerne alle regions
- Vise at et program med region annoteringer bevarer betydning af erasure udgaven af programmet
- Kan vise en betinget korrekthed:  
Lad  $t$  være et regions annoteret program og antag at  $eval_R(t) \neq wrong$ . Så gælder der at  $eval_R(t) = eval(\|t\|)$  annoteringerne.



## Annotering af programmer - 2

Bevis ved hjælp af en række lemmaer.

NB: det er ikke muligt med given syntax at kende forskel på regions "wrong" og alm. typefejl "wrong" men det er efter sigende en ligetil udvidelse.

- Lemma over almost values og alm. values stadig er værdier
- Simulation
- Simulated progress
- Et lemma for hver af  $eval_R(t) = \perp$  og  $eval_R(t) = bv$



## Annotering af programmer - 3

Antag at  $t$  er en værdi  $v$  eller en allmost-value  $u$ . Så er  $||t||$  en værdi i BL

Bevis ved strukturel induktion over  $t$ .

Bruger induktions hypotese ved region abstractioner og closures.



## Annotering af programmer - 4

Simulation lemma:

Antag  $t \rightarrow^{RAL} t'$ . Så gælder det enten (a)  $\|t\| \rightarrow \|t'\|$  eller (b)  $\|t\| = \|t'\|$ .

Induktion over afledning af  $t \rightarrow^{RAL} t'$



## Annotering af programmer - 5

Simulated progress:

Antag  $t \rightarrow^{RAL} t'$  men ikke  $\|t\| \rightarrow \|t'\|$ . Så er  $t'$  strengt mindre end  $t$  under en målestok hvor ikke-evaluerede abstraktioner er "større" end closures.

Bevis(oplæg):

Antallet af kontekst regler der skal bruges fåes ud fra simulation lemmaet. RE-CLOS og RE-RCLOS opfyldes ud fra definitionen af målestokken.

Fra RE-BETA og RE-DEALLOC, fåes det at region substitution ikke ændrer på størrelsen men at reduktionerne enten fjerner  $\lambda$  eller  $\text{new}$ .



## Annotering af programmer - 6

Antag  $\text{eval}_R(t) = bv$ . Så gælder det at  $\text{eval}(\|t\|) = bv$

Bevis:

Fra definitionen af  $\text{eval}_R$  har vi at  $t \rightarrow^{*!} bv$ . Ved hjælp af simulation lemmaet på reduktions skridtene fåes det at  $\|t\| \rightarrow^{BL*} \|bv\| = bv$   
Og da  $bv$  ikke har nogen efterfølger gælder det at  $\text{eval}(\|t\|) = bv$



# Annotering af programmer - 7

Antag  $\text{eval}_R(t) = \perp$ . Så gælder det at  $\text{eval}(\|t\|) = \perp$

Bevis:

Antagelsen siger at der er uendelig mange reduktioner

$t_0 \rightarrow^{RAL} t_1 \rightarrow^{RAL} \dots \rightarrow^{RAL} t_i \dots$ . Simulation lemma siger at for  $i \geq 0$  at enten  $\|t_i\| = \|t_{i+1}\|$  eller  $\|t_i\| \rightarrow^{BL} \|t_{i+1}\|$

Simulated progress sikrer at der ikke eksisterer et  $N$  således at  $\|t_i\| \rightarrow \|t_{i+1}\|$  ikke kan lade sig gøre for all  $i > N$ .

Derfor kan man vælge  $i$ 'er så der også er uendeligt mange BL reduktioner. Og derfor  $\text{eval}(\|t_0\|) = \perp$ .

# Agenda

- 1 Valueflow vha label-typning
  - Tagget sprog
  - Flowinformation vha. labels
  - Korrekthed
  - Inferens
- 2 Effekter
  - Scoped tagget sprog (STL)
  - Scoped effect typet sprog (ETL)
- 3 Regionsbaseret lagerhåndtering
  - Region-Annotated sprog (RAL)
  - Annotering af programmer
- 4 Tofte-Talpin typesystemet
  - Sundhed



# Indledning

Vores egen Mads Tofte har sammen med Jean-Pierre Talpin lavet et typesystem, som i en let omformet udgaver er typesystemet RTL for RAL.

- Typen  $\forall X. T$  binder typevariablen  $X$  i  $T$
- Typen  $\Pi \rho. \varphi T$  binder regions variabelen  $\rho$  i  $T$  og  $\varphi$
- Typen  $\forall \epsilon. T$  binder effect variabelen  $\epsilon$  i  $T$ .
- RTL er effect og type polymorft, men i modsætning til System F er det uden explicit syntax til at introducere og eliminere.

# Sundhed

Her skal vi så vise at veltypede programmer også er lager-sikre med RTL.

Hjælpe lemmaer omkring formen på  $\Gamma$  og hvilke regler afledningen af en term kan slutte med.

- 1 Hvis  $\Gamma \vdash t :^{\varphi} T$ ,  $dom(\Gamma') = fv(t)$  og  $\Gamma$  og  $\Gamma'$  er enige når begge er definerede så  $\Gamma' \vdash t :^{\varphi} T$ .
- 2 Lad  $S$  være en erstatning af kendt form. Hvis  $\Gamma \vdash t :^{\varphi} T$  kan afledes i  $n$  skridt så kan  $S\Gamma \vdash St :^{S\varphi} ST$  også afledes i  $n$  skridt.
- 3 Antag at  $\Gamma \vdash v :^{\varphi} T$  har en afledning i  $n$  skridt. Så har den en afledning i max.  $n$  skridt hvor den sidste regl der er brugt ikke er RT-TInst eller RT-EInst.
- 4 Et canonical forms lemma



## Sundhed - 2

Derefter skal der bruges nogle lemmaer omkring effekter

① Enlarge:

Hvis  $\Gamma \vdash t : \varphi \ T$  og  $\varphi \subseteq \varphi'$  så gælder der  $\Gamma \vdash t : \varphi' \ T$

② Value-No-Effect:

Lad  $v$  være en værdi. Hvis  $\Gamma \vdash v : \varphi' \ T$  så gælder der  $\Gamma \vdash t : \emptyset \ T$

## Sundhed - 3

Derefter kan vi bevise de nødvendige lemmaer for at lave sundheds-beviset.

- Substitution
- Subject reduction (preservation)
- Progress



## Sundhed - 4

Substitution:

Hvis  $\Gamma, x_1 : T_1 \vdash t :^{\varphi} T$  og  $\Gamma \vdash t_1 :^{\circ} T$  så gælder der at  
 $[x_1 \mapsto t_1]t :^{\varphi} T$ .

Bevis:

Induktion over type afledning for t.



## Sundhed - 5

Preservation:

Hvis  $\Gamma \vdash t :_{\varphi} T$  og  $t \rightarrow^{RAL} t'$  så gælder der  $\Gamma \vdash t' :_{\varphi} T$

Bevis:

Induktion over type-afledning.



## Sundhed - 6

Progress:

Hvis  $\circlearrowleft \vdash t :^\varphi T$  og  $\bullet \notin \varphi$  så er  $t$  enten en værdi eller der eksisterer et  $t'$  så  $t \rightarrow^{RAL} t'$ .

Bevis:

Induktion over typeafledning af  $\circlearrowleft \vdash t :^\varphi T$ .

