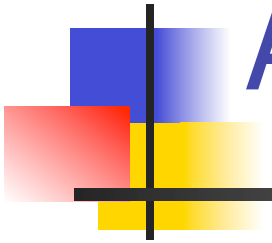
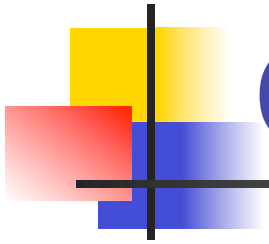


# Effect types and region-based memory management: A gentle introduction

Fritz Henglein, DIKU

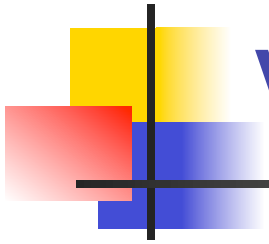
2009-08-13

Based on Henglein, Makholm, Niss, "Effect type systems and region-based memory management," in Advanced Topics in Types and Programming Languages, Benjamin Pierce (ed.), MIT Press, 2005



# Overview

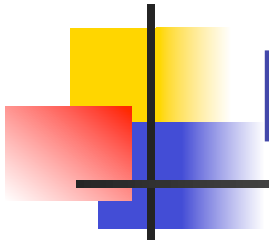
- Value flow analysis revisited
- Region typing: Value flow analysis reinterpreted
- Region inference: Trivial and principal completions
- Scoped regions
- Type and effect system
- Region and effect polymorphism
- Extensions (references)



# Value flow analysis

- Figuring out where values are *created* and where those are *destructed.*
- Why?
  - Software understanding (e.g.,error reporting)
  - Optimization (e.g., constant folding)
  - Interpretation (e.g., binding-time analysis, dynamic/soft typing)

# Value flow analysis: Basic problems



- Not a well-defined problem: Exact solution for Turing complete languages is impossible (Rice's Theorem)
- Which *specific* value flow analysis problem then?
- Ensuring correctness vis a vis operational semantics
- Representing and exploiting result of value flow analysis

*Terms*

| | | |
|---|---|---|
| $t$ ::= | | *terms:* |
| | $v$ | *value expression* |
| | $x$ | *variable* |
| | $t\ t$ | *application* |
| | $\texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t$ | *conditional* |
| | $\texttt{fix}\ x.t$ | *recursion* |
| $v$ ::= | | *value expressions:* |
| | $\lambda x.t$ | *abstraction* |
| | $bv$ | *truth value* |
| $bv$ ::= | | *truth values:* |
| | $\texttt{tt}$ | *true* |
| | $\texttt{ff}$ | *false* |

*Evaluation rules* $\boxed{t \longrightarrow t'}$

$$(\lambda x.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-BETA)}$$

$$\texttt{fix}\ x.t \longrightarrow [x \mapsto \texttt{fix}\ x.t]t \quad \text{(E-FIXBETA)}$$

$$\texttt{if tt then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \quad \text{(E-IFTRUE)}$$

$$\texttt{if ff then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \quad \text{(E-IFFALSE)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-APP1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-APP2)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \quad \text{(E-IF)}$$

*Types*

| | | |
|---|---|---|
| $T$ ::= | | *types:* |
| | $\texttt{bool}$ | *Boolean type* |
| | $T \to T$ | *function type* |

*Typing rules* $\boxed{\Gamma \vdash t : T}$

$$\frac{x \notin \Gamma'}{\Gamma, x : T, \Gamma' \vdash x : T} \quad \text{(T-VAR)}$$

$$\Gamma \vdash bv : \texttt{bool} \quad \text{(T-BOOL)}$$

$$\frac{\Gamma \vdash t_1 : \texttt{bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T} \quad \text{(T-IF)}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x.t : T_1 \to T_2} \quad \text{(T-ABS)}$$

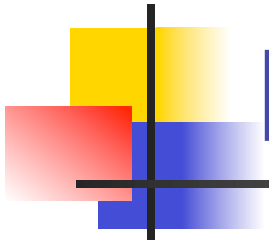
$$\frac{\Gamma \vdash t_0 : T_1 \to T_2 \qquad \Gamma \vdash t_1 : T_1}{\Gamma \vdash t_0\ t_1 : T_2} \quad \text{(T-APP)}$$

$$\frac{\Gamma, x : T \vdash t : T}{\Gamma \vdash \texttt{fix}\ x.t : T} \quad \text{(T-FIX)}$$

*Derived form*

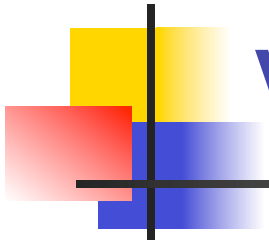
$$\texttt{let}\ x = t_1\ \texttt{in}\ t_2 \stackrel{\text{def}}{=} (\lambda x.t_2)\ t_1$$

**Figure 1-1: Base language BL.**



# Basic notions:

- A closed term t is *final* if it cannot be reduced.
  - All values are final.
  - All other final terms are called *stuck terms (*or stuck states); they signal the occurrence of a (type) error, e.g., *true false.*
- t →* v: Closed term t *evaluates* to value v.

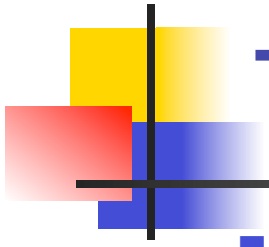


# Value-flow analysis: Example

EXAMPLE: Consider the BL-program $t_0$:

```
let fst= λu.λv.u in
    (let x = λp.p tt ff in λy.λq.q (x fst) y)
    tt
```

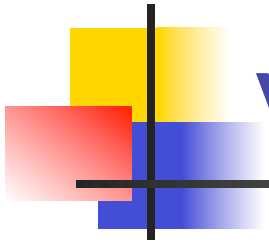
Value flow analysis should tell us that x may be applied to fst (which is rather easy to see), fst may be applied to tt (which is not immediately obvious from the source code), and the λ-abstraction λy.λp.p (x fst) y may be applied to tt, but λp.p (x fst) y is not applied anywhere.



# Typed instrumentation

- Instrument source language with annotations that admit the intended observations (*completion* of underlying program)
- Provide operational semantics for annotations
- Extend static semantics (type system) to annotations
- Show that annotated language preserves underlying source language semantics:
  - *Partial correctness*: Any completion gives same result as underlying program or goes wrong
  - *Soundness*: Any well-typed completion does not go wrong (does not get stuck).
  - *Corollary*: Any well-typed completion is correct.

# Typed instrumentation for value flow analysis



- Tags (labels): $p_1$, $p_2$, ...
- Tagging operation: *t at p*
- Untagging operation: *t ! p*
- Intuition:
  - *t at p*: names "the value" of t, where t is usually a value (constructor) expression (in BL: true, false or a λ-abstraction)
  - *t ! p*: expresses that t evaluates to a value with name p is used here, where t is usually a term in destructive context (in BL: if [] then t1 else t2, [] t)

*New terms*

| | | | |
|---|---|---|---|
| t | ::= | ... | *terms:* |
| | | t at p | *tagging* |
| | | t ! p | *untagging* |
| v | ::= | ... | *value expressions:* |
| | | $\langle v \rangle_p$ | *tagged value* |
| p | ::= | | *label expressions:* |
| | | $\rho$ | *label variable* |

*New evaluation rules* $\boxed{t \longrightarrow t'}$

$$\frac{t \longrightarrow t'}{t \text{ at } \rho \longrightarrow t' \text{ at } \rho} \quad \text{(E-TAG)}$$

$$v \text{ at } \rho \longrightarrow \langle v \rangle_\rho \quad \text{(E-TAGBETA)}$$

$$\frac{t \longrightarrow t'}{t \text{ ! } \rho \longrightarrow t' \text{ ! } \rho} \quad \text{(E-UNTAG)}$$

$$\langle v \rangle_\rho \text{ ! } \rho \longrightarrow v \quad \text{(E-UNTAGBETA)}$$

*New types*

| | | | |
|---|---|---|---|
| T | ::= | ... | *types:* |
| | | T at p | *tagged value type* |

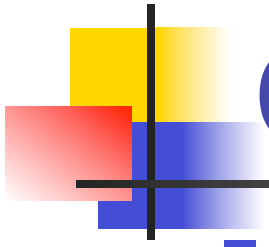
*New typing rules* $\boxed{\Gamma \vdash t : T}$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t \text{ at } p : T \text{ at } p} \quad \text{(T-TAG)}$$

$$\frac{\Gamma \vdash t : T \text{ at } p}{\Gamma \vdash t \text{ ! } p : T} \quad \text{(T-UNTAG)}$$

$$\frac{\Gamma \vdash v : T}{\Gamma \vdash \langle v \rangle_p : T \text{ at } p} \quad \text{(T-TAGVALUE)}$$

**Figure 1-2: Tagged Language, TL (Extension of BL)**



# Observations

- The operational semantics consists of:
  - the evaluation rules for BL (T1 rules), plus
  - evaluation rules for the new constructs (T2 rules)
- The type system consists of:
  - the type rules for BL
  - type rules for the new constructs

# Erasures and completions

DEFINITION [ERASURE, COMPLETION]: Let $\mathtt{t} \in \mathrm{TL}$. Then the *erasure* $\|\mathtt{t}\|$ of term $\mathtt{t}$ is defined as follows:

$$
\begin{aligned}
\|\mathtt{x}\| &= \mathtt{x} \\
\|\mathtt{t}_1\ \mathtt{t}_2\| &= \|\mathtt{t}_1\|\ \|\mathtt{t}_2\| \\
\|\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3\| &= \mathtt{if}\ \|\mathtt{t}_1\|\ \mathtt{then}\ \|\mathtt{t}_2\|\ \mathtt{else}\ \|\mathtt{t}_3\| \\
\|\mathtt{fix}\,\mathtt{x}.\mathtt{t}\| &= \mathtt{fix}\,\mathtt{x}.\|\mathtt{t}\| \\
\|\lambda\mathtt{x}.\mathtt{t}\| &= \lambda\mathtt{x}.\|\mathtt{t}\| \\
\|\mathtt{tt}\| &= \mathtt{tt} \\
\|\mathtt{ff}\| &= \mathtt{ff} \\
\|\mathtt{t}\ \mathtt{at}\ \mathtt{p}\| &= \|\mathtt{t}\| \\
\|\mathtt{t}\ !\ \mathtt{p}\| &= \|\mathtt{t}\| \\
\|\langle \mathtt{v} \rangle_{\mathtt{p}}\| &= \|\mathtt{v}\|
\end{aligned}
$$

Conversely, we call a TL-term $\mathtt{t}'$ a *completion* of BL-term $\mathtt{t}$ if $\|\mathtt{t}'\| = \mathtt{t}$. □

# Con/decon completions

Con/decon completions ensure that each constructor is tagged and each destructive position is untagged, and nothing else.

*Con/decon completion templates*

| | | | |
|---|---|---|---|
| t | ::= | | : |
| | | v | |
| | | x | |
| | | (t ! p) t | |
| | | if (t ! p) then t else t | |
| | | fix x.t | |
| v | ::= | | : |
| | | (λx.t) at p | *abstraction* |
| | | bv at p | *truth value* |
| bv | ::= | | *truth values:* |
| | | tt | *true* |
| | | ff | *false* |

**Figure 1-3: Con/decon completions**

# Value-flow analysis: Example continued

EXAMPLE: Consider the BL-program $\mathtt{t}_0$:

$$\begin{array}{l}\mathtt{let\ fst=\ } \lambda \mathtt{u}.\lambda \mathtt{v}.\mathtt{u}\ \mathtt{in}\\ \quad (\mathtt{let\ x} = \lambda \mathtt{p}.\mathtt{p\ tt\ ff\ in}\ \lambda \mathtt{y}.\lambda \mathtt{q}.\mathtt{q}\ (\mathtt{x\ fst})\ \mathtt{y})\\ \quad \mathtt{tt}\end{array}$$

The following con/decon completion $\mathtt{t}_1$ of $\mathtt{t}_0$ captures this:

$$\begin{array}{l}\mathtt{let\ fst=\ } \lambda_{l_K} \mathtt{u}.\lambda_{l_b} \mathtt{v}.\mathtt{u}\ \mathtt{in}\\ \quad (\mathtt{let\ x} = \lambda_{l_x} \mathtt{p}.((\mathtt{p}^{l_K}\ \mathtt{tt}_{l_t})^{l_b}\ \mathtt{ff}_{l_f})\ \mathtt{in}\\ \quad\ \ \lambda_{l_f} \mathtt{y}.\lambda_{l_c} \mathtt{q}.((\mathtt{q}^{l_q}\ (\mathtt{x}^{l_x}\ \mathtt{fst}))^{l_d}\ \mathtt{y}))\\ \quad \mathtt{tt}_{l_t}\end{array}$$

To make the completion more readable, we have written $\lambda_{\mathtt{p}}\mathtt{x}.\mathtt{t}$ for $(\lambda \mathtt{x}.\mathtt{t})$ at p, $\mathtt{bv}_{\mathtt{p}}$ for bv at p, and $(\mathtt{t}^{\mathtt{p}}\ \mathtt{t}')$ for $(\mathtt{t}\ !\ \mathtt{p})\ \mathtt{t}'$. □

# Correctness Theorem

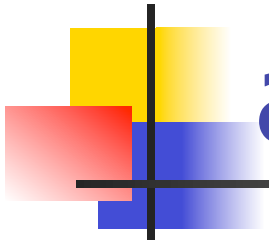
Any well-typed completion of a BL-term computes the same result as the BL-term itself:

COROLLARY [CORRECTNESS]: Let $\mathtt{t}$ be a closed TL-term and $\mathtt{v}$ a TL-value.

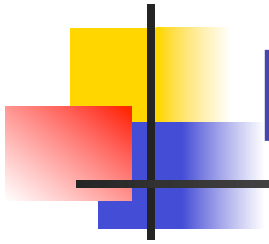
1. $\mathtt{t} \uparrow$ if and only if $\|\mathtt{t}\| \uparrow$.

2. $\|\mathtt{t}\| \xrightarrow{\mathrm{BL}}{}^{*} \|\mathtt{v}\|$ if and only if there exists a TL-value $\mathtt{v}'$ such that $\|\mathtt{v}'\| = \|\mathtt{v}\|$ and $\mathtt{t} \xrightarrow{\mathrm{T}}{}^{*} \mathtt{v}'$. □

# Partial correctness and soundness



- Correctness is a corollary of:
  - Partial correctness: Any completion computes the same result or goes wrong
  - Soundness: No well-typed completion goes wrong.
- Note:
  - Partial correctness is independent of the type system; it is a property of the evaluation rules (instrumented operational semantics) alone.
  - Soundness is a property of the type system. It doesn't say anything about the relation of the computed result to the result of the underlying term (its erasure).



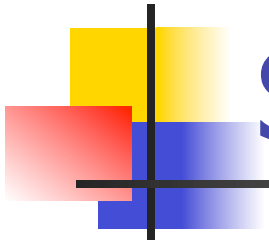
# Partial correctness

LEMMA [SIMULATION]: Let $t, t_1, t_2$ range over TL-terms.

1. If $v$ is a value expression then so is $\|v\|$.
2. $\xrightarrow{T_2}$ is strongly normalizing.
3. If $t_1 \xrightarrow{T_1} t_2$ then $\|t_1\| \xrightarrow{BL} \|t_2\|$.
4. If $t_1 \xrightarrow{T_2} t_2$ then $\|t_1\| = \|t_2\|$.

THEOREM [CONDITIONAL CORRECTNESS]: For TL-terms $t, t'$ we have:

1. If $t \xrightarrow{T}{}^{*} t'$ then $\|t\| \xrightarrow{BL}{}^{*} \|t'\|$.
2. If $t \uparrow$ then $\|t\| \uparrow$.
3. If $\|t\|$ gets stuck then $t$ gets stuck, too. □



# Soundness

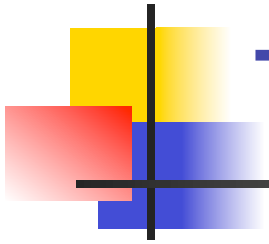
Soundness is proved by showing that

- a well-typed term cannot be stuck (*progress*)
- well-typedness is preserved under reduction (*preservation)*

LEMMA [SUBJECT REDUCTION (PRESERVATION)]: Let $\mathtt{t}, \mathtt{t}'$ be TL-terms. If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \xrightarrow{\mathtt{T}} \mathtt{t}'$ then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$. □

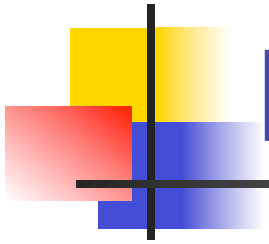
LEMMA [PROGRESS]: If $\vdash \mathtt{t} : \mathtt{T}$ then either $\mathtt{t} = \mathtt{v}$ for some value (closed value expression) $\mathtt{v}$ or there exists $\mathtt{t}'$ such that $\mathtt{t} \xrightarrow{\mathtt{T}} \mathtt{t}'$. □

THEOREM [SOUNDNESS]: If $\vdash \mathtt{t} : \mathtt{T}$ then evaluation of $\mathtt{t}$ does not get stuck. □



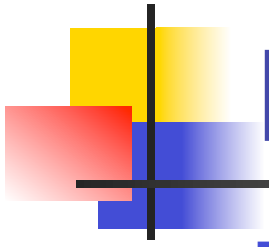
# Triviality and Principality

- Any well-typed completion represents *correct* value flow information, but does it always represent *useful* value flow information?
  - The *trivial* con/decon completion, which tags/untags with a single label $p_h$, provides *no* useful (new) information; it is basically just a trivial *embedding* of the underlying term into the language of con/decon completions.
- Is there a *principal completion?* That is, one whose value flow information subsumes the value flow information of any other completion of the same term?



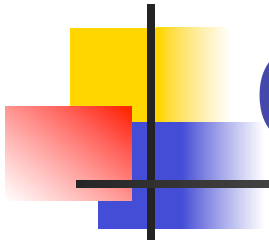
# Principal completions

- Definition:
  - Completion t *subsumes* t' if there exists a substitution *S* on labels (only) such that S(t) = t'.
  - Completion t is *principal* for ||t|| if it subsumes all completions of ||t||.
- Theorem: Each ||t|| has a principal completion.



# Notes:

- The value flow analysis captured by principal completions here corresponds to Simple Value Flow Analysis (equational value flow analysis).
- Monovariant VFA (aka 0CFA) can be captured by extended the label language with disjunctions (label sets) with attendant subtyping rules.
- SVFA and MonoVFA can be extended to polymorphic VFA.

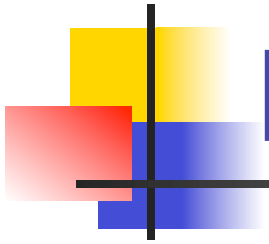


# Observations

- Value flow information has an *operational semantics* of its own: it can be executed!
- The *instrumented language* (annotations and operational semantics) captures 'exactly' value flow analysis: a completion of a closed term v of type bool contains *semantically correct* value flow information if its evaluation does not go wrong.
- The *type system* limits correct completions to a well-specified set of correct completions, for each of which it is feasible to verify its correctness.

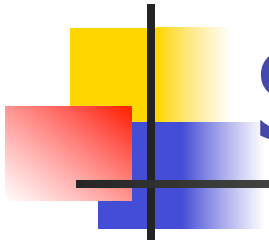
# From VFA to region-based memory management

- Recall that well-typed completions with tagging and untagging operations represent correct value flow information
- Idea:
  - Reinterpret labels as *regions:* a region is a chunk of extendible memory
  - Reinterpret *t at p*: allocate the value of *t* in region *p* and return pointer to value.
  - Reinterpret *t ! p*: Check that *t* evaluates to a pointer into *p* and fetch the value for it. Note: The check can be elided---it is alwas true---for well-typed completions.



# Problem: Global regions!

- Tags are global.
- Value flow information does not say anything when regions come into existence (get allocated) and when they cease to exist (get deallocated and underlying memory recycled)
- Need to do *lifetime analysis* for regions and *represent result* of analysis operationally.



# Scoped regions: Basic idea

- Consider a value flow judgement $\Gamma$ |- t: T and consider a region (tag) p that occurs in t. Then, *intuitively*,
  - If p does not occur in $\Gamma$ then the environment in which t evaluates contains no values in p.
  - If p does not occur in T then no values stored in p are returned to the context of t.
  - So p can be *allocated* before evaluation of t, *accessed* during evaluation of t, and then *deallocated* upon termination of evaluation of t.
- Introduce region-scoped term: *new p. t*

# Naively region-scoped TL

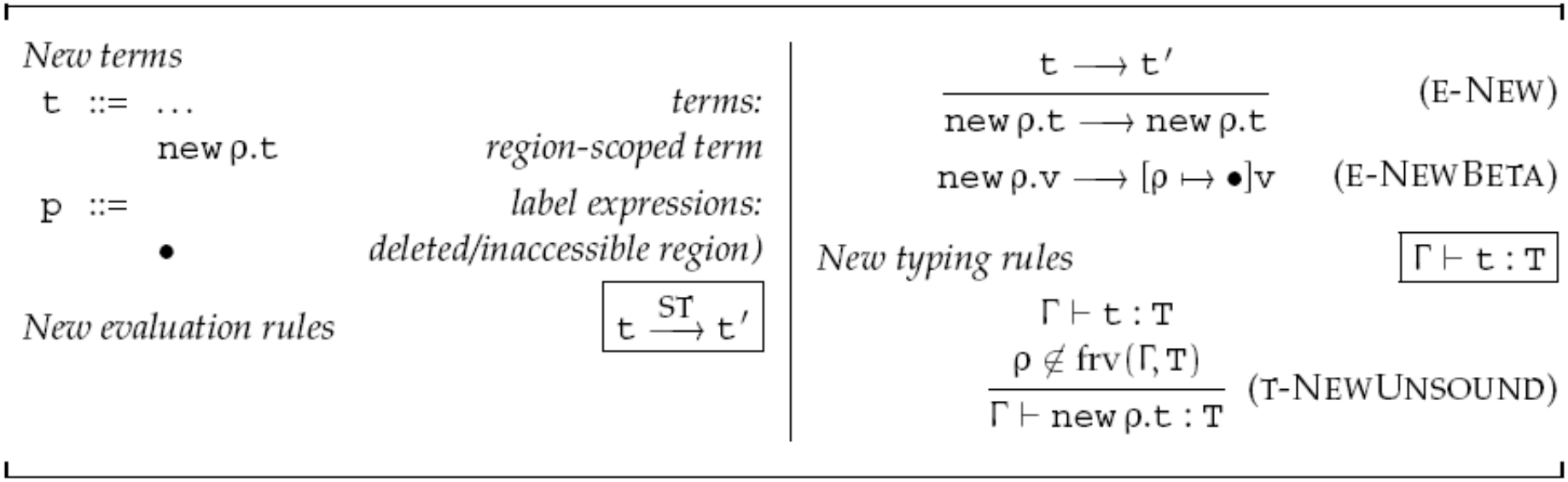



**Figure 1-4: Scoped Tagged Language (unsound), STL (Basis: TL)**

# Problem: Unsoundness!

EXAMPLE: Consider the following STL-term $\mathtt{t_f} =$:

$$\mathtt{new}\,\rho_0.\mathtt{let}\,\mathtt{x} = \mathtt{tt}\,\mathtt{at}\,\rho_0\,\mathtt{in}\,\lambda\mathtt{y}.\mathtt{if}\,\mathtt{x}\,!\,\rho_0\,\mathtt{then}\,\mathtt{y}\,\mathtt{else}\,\mathtt{ff}\,\mathtt{at}\,\rho_1$$

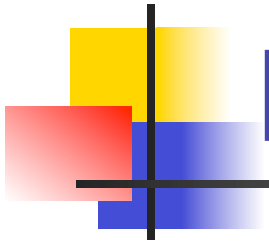
It reduces as follows:

$$\begin{array}{ll}
\mathtt{new}\,\rho_0.\mathtt{let}\,\mathtt{x} = \mathtt{tt}\,\mathtt{at}\,\rho_0\,\mathtt{in}\,\lambda\mathtt{y}.\mathtt{if}\,\mathtt{x}\,!\,\rho_0\,\mathtt{then}\,\mathtt{y}\,\mathtt{else}\,\mathtt{ff}\,\mathtt{at}\,\rho_1 & \longrightarrow \\
\mathtt{new}\,\rho_0.\mathtt{let}\,\mathtt{x} = \langle\mathtt{tt}\rangle_{\rho_0}\,\mathtt{in}\,\lambda\mathtt{y}.\mathtt{if}\,\mathtt{x}\,!\,\rho_0\,\mathtt{then}\,\mathtt{y}\,\mathtt{else}\,\mathtt{ff}\,\mathtt{at}\,\rho_1 & \longrightarrow \\
\mathtt{new}\,\rho_0.\lambda\mathtt{y}.\mathtt{if}\,\langle\mathtt{tt}\rangle_{\rho_0}\,!\,\rho_0\,\mathtt{then}\,\mathtt{y}\,\mathtt{else}\,\mathtt{ff}\,\mathtt{at}\,\rho_1 & \longrightarrow \\
\lambda\mathtt{y}.\mathtt{if}\,\langle\mathtt{tt}\rangle_{\bullet}\,!\,\bullet\,\mathtt{then}\,\mathtt{y}\,\mathtt{else}\,\mathtt{ff}\,\mathtt{at}\,\rho_1 &
\end{array}$$

$$\lambda\mathtt{y}.\mathtt{if}\,\langle\mathtt{tt}\rangle_{\bullet}\,!\,\bullet\,\mathtt{then}\,\mathtt{y}\,\mathtt{else}\,\mathtt{ff}\,\mathtt{at}\,\rho_1$$

is a value; it is *not* stuck. It is easy to see, however, how it can give rise to a stuck state. The program $\mathtt{t_f}\,(\mathtt{tt}\,\mathtt{at}\,\rho_1)$ is a well-typed STL-program of type $\mathtt{bool}\,\mathtt{at}\,\rho_1$, yet evaluation gets stuck:

$$\begin{array}{ll}
\mathtt{t_f}\,(\mathtt{tt}\,\mathtt{at}\,\rho_1) & \stackrel{*}{\longrightarrow} \\
(\lambda\mathtt{y}.\mathtt{if}\,\langle\mathtt{tt}\rangle_{\bullet}\,!\,\bullet\,\mathtt{then}\,\mathtt{y}\,\mathtt{else}\,\mathtt{ff}\,\mathtt{at}\,\rho_1)\,(\mathtt{tt}\,\mathtt{at}\,\rho_1) & \longrightarrow \\
\mathtt{if}\,\langle\mathtt{tt}\rangle_{\bullet}\,!\,\bullet\,\mathtt{then}\,\mathtt{ff}\,\mathtt{at}\,\rho_1\,\mathtt{else}\,\mathtt{ff}\,\mathtt{at}\,\rho_1 &
\end{array}$$



# Effects

- Type of lexical closures does not include mention types of environment part
- Environment part may be accessed after closure is computed.
- *Idea*:
  - Capture not only *type* of result value of lexical closure, but also *effect* of its computation on environment part.

# Effect type judgements

The basic effect type judgement is

$$\Gamma \vdash t :^{\varphi} T$$

where $\varphi$ is an *effect expression* (henceforth simply called *effect*) and ${}^{\varphi}T$ is an *effect type* or *type and effect*. The judgement should be read informally as "Under the assumptions $\Gamma$, the evaluation of $t$ may have the observable effect $\varphi$, and it eventually yields a value of type $T$, if any." For program analysis purposes *observable* may also be understood as *interesting*. When an evaluation has no observable effect, we say it has the *empty effect*, written $\emptyset$, and ${}^{\emptyset}T$ is abbreviated to $T$.

For RBMM: effect = the regions (possibly) accessed during evaluation.

*Terms*

| | | | |
|---|---|---|---|
| t | ::= | | *terms:* |
| | | v | *value expression* |
| | | x | *variable* |
| | | t t | *application* |
| | | if t then t else t | *conditional* |
| | | t at p | *tagging* |
| | | t ! p | *untagging* |
| | | new ρ.t | *label-scoped term* |
| | | fix x.t | *recursion* |
| v | ::= | | *value expressions:* |
| | | λx.t | *abstraction* |
| | | bv | *truth value* |
| | | $\langle v \rangle_p$ | *tagged value* |
| bv | ::= | | *truth values:* |
| | | tt | *true* |
| | | ff | *false* |
| p | ::= | | *label/region expressions:* |
| | | ρ | *label/region variable)* |
| | | • | *deleted/inaccessible label/region)* |

*Effect expressions*

| | | | |
|---|---|---|---|
| φ | ::= | {ρ, . . . , ρ} | *effect expressions:* |

*Types*

| | | | |
|---|---|---|---|
| T | ::= | | *types:* |
| | | bool | *Boolean type* |
| | | $T \to {}^{\varphi} T$ | *function type* |
| | | T at p | *tagged value type* |

*Effect typing rules* $\boxed{\Gamma \vdash t :^{\varphi} T}$

$$\frac{x \notin \Gamma'}{\Gamma, x : T, \Gamma' \vdash x :^{\varphi} T} \quad \text{(TE-VAR)}$$

$$\Gamma \vdash bv :^{\varphi} \text{bool} \quad \text{(TE-BOOL)}$$

$$\frac{\Gamma \vdash t_1 :^{\varphi} \text{bool} \qquad \Gamma \vdash t_2 :^{\varphi} T \qquad \Gamma \vdash t_3 :^{\varphi} T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 :^{\varphi} T} \quad \text{(TE-IF)}$$

$$\frac{\Gamma, x : T_1 \vdash t :^{\varphi_2} T_2}{\Gamma \vdash \lambda x.t :^{\varphi_1} T_1 \to {}^{\varphi_2} T_2} \quad \text{(TE-ABS)}$$

$$\frac{\Gamma \vdash t_0 :^{\varphi} T_1 \to {}^{\varphi} T_2 \qquad \Gamma \vdash t_1 :^{\varphi} T_1}{\Gamma \vdash t_0\ t_1 :^{\varphi} T_2} \quad \text{(TE-APP)}$$

$$\frac{\Gamma \vdash t :^{\varphi} T \qquad p \in \varphi}{\Gamma \vdash t \text{ at } p :^{\varphi} T \text{ at } p} \quad \text{(TE-AT)}$$

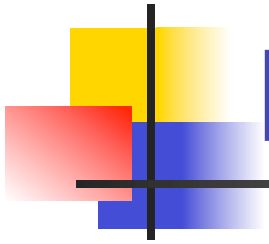
$$\frac{\Gamma \vdash t :^{\varphi} T \text{ at } p \qquad p \in \varphi}{\Gamma \vdash t \,!\, p :^{\varphi} T} \quad \text{(TE-FROM)}$$

$$\frac{\Gamma \vdash v :^{\varphi} T}{\Gamma \vdash \langle v \rangle_p :^{\varphi} T \text{ at } p} \quad \text{(TE-CELL)}$$

$$\frac{\Gamma \vdash t :^{\varphi} T \qquad \rho \notin \text{frv}(\Gamma, T)}{\Gamma \vdash \text{new } \rho.t :^{\varphi - \{\rho\}} T} \quad \text{(TE-NEW)}$$

$$\frac{\Gamma, x : T \vdash t :^{\varphi} T}{\Gamma \vdash \text{fix } x.t :^{\varphi} T} \quad \text{(TE-FIX)}$$

**Figure 1-5: Scoped effect typed language ETL(sound).**



# Example reconsidered

EXAMPLE: Consider the term

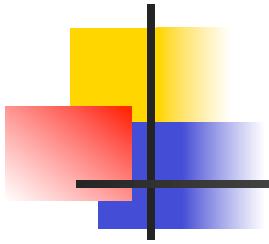
$$\mathtt{t_f} = \mathtt{new}\,\rho_0.\mathtt{let\ x} = \mathtt{tt\ at}\ \rho_0\ \mathtt{in}\ \lambda \mathtt{y.if\ x}\,!\,\rho_0\ \mathtt{then\ y\ else\ ff\ at}\ \rho_1$$

from Example 1.2.20 again. Whereas it is typable in STL even though it gets stuck when applied to an argument, it is not typable in ETL. To see this, consider the let-expression $\mathtt{t_l}$

$$\mathtt{let\ x} = \mathtt{tt\ at}\ \rho_0\ \mathtt{in}\ \lambda \mathtt{y.if\ x}\,!\,\rho_0\ \mathtt{then\ y\ else\ ff\ at}\ \rho_1$$

inside $\mathtt{t_f}$. Its ETL effect type $\mathtt{T_l}$ is $^{\{\rho_0\}}(\mathtt{bool\ at}\ \rho_1 \rightarrow {}^{\{\rho_0\}}\mathtt{bool\ at}\ \rho_1)$. Note that $\rho_0$ occurs in the effect, but neither in the function type's domain nor its range type. This reflects the fact that an application of $\mathtt{t_l}$ may access region $\rho_0$. Since $\rho_0 \in \mathrm{frv}(\mathtt{bool\ at}\ \rho_1 \rightarrow {}^{\{\rho_0\}}\mathtt{bool\ at}\ \rho_1)$ Rule (TE-NEW) is *not* applicable, and so there is no way of inferring a type for $\mathtt{t_f}$, which indeed would be unsound. □

# Region and effect polymorphism



- Monomorphic RBMM has very limited practical utility since it does not provide context indepence for function calls:
  - multiple calls to the same function require that respective arguments and results are put into same region.
- Introduce region polymorphism: Regions may be parameters to functions
- Polymorphic RBMM with a monomorphic recursion rule is of limited practical utility:
  - (multiple) recursive calls put values in same region.
- Introduce polymorphic recursion.

*Terms*

| | | |
|---|---|---|
| $t$ ::= | | *terms:* |
| | $u$ | *value or almost value* |
| | $x$ | *variable* |
| | $\texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t$ | *conditional* |
| | $\texttt{fix}\ x.u$ | *recursion* |
| | $t\ t$ | *application* |
| | $t\ [\![p]\!]$ | *region app.* |
| | $\texttt{new}\ \rho.t$ | *letregion* |
| $u$ ::= | | *almost values:* |
| | $v$ | *value* |
| | $(\lambda x.t)\ \texttt{at}\ p$ | *abstraction* |
| | $(\lambda \rho.u)\ \texttt{at}\ p$ | *region abs.* |
| $v$ ::= | | *value expressions:* |
| | $bv$ | *truth value* |
| | $\langle \lambda x.t \rangle_p$ | *closure* |
| | $\langle \lambda \rho.u \rangle_p$ | *region closure* |
| $bv$ ::= | | *truth values:* |
| | $\texttt{tt}$ | *true* |
| | $\texttt{ff}$ | *false* |
| $p$ ::= | | *places:* |
| | $\rho$ | *region variable* |
| | $\bullet$ | *deallocated* |

*Evaluation* $\boxed{t \xrightarrow{\text{RAL}} t'}$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \longrightarrow \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3} \quad \text{(RE-IF)}$$

$$\texttt{if tt then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_2 \quad \text{(RE-IFTRUE)}$$

$$\texttt{if ff then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_3 \quad \text{(RE-IFFALSE)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(RE-APP1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(RE-APP2)}$$

$$\lambda x.t\ \texttt{at}\ \rho \longrightarrow \langle \lambda x.t \rangle_\rho \quad \text{(RE-CLOS)}$$

$$\langle \lambda x.t \rangle_\rho\ v \longrightarrow [x \mapsto v]t \quad \text{(RE-BETA)}$$

$$\frac{u \longrightarrow u'}{\texttt{fix}\ x.u \longrightarrow \texttt{fix}\ x.u'} \quad \text{(RE-FIX)}$$

$$\texttt{fix}\ x.v \longrightarrow [x \mapsto \texttt{fix}\ x.v]v \quad \text{(RE-FIXBETA)}$$

$$\frac{t \longrightarrow t'}{t\ [\![p]\!] \longrightarrow t'\ [\![p]\!]} \quad \text{(RE-RAPP)}$$

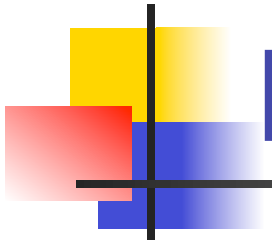
$$\lambda \rho_1.u\ \texttt{at}\ \rho_2 \longrightarrow \langle \lambda \rho_1.u \rangle_{\rho_2} \quad \text{(RE-RCLOS)}$$

$$\langle \lambda \rho_1.u \rangle_{\rho_2}\ [\![p]\!] \longrightarrow [\rho_1 \mapsto p]u \quad \text{(RE-RBETA)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{new}\ \rho.t_1 \longrightarrow \texttt{new}\ \rho.t_1'} \quad \text{(RE-LETREG)}$$

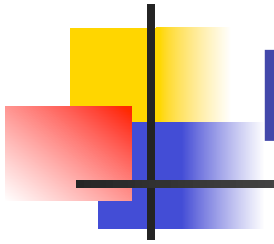
$$\texttt{new}\ \rho.v \longrightarrow [\rho \mapsto \bullet]v \quad \text{(RE-DEALLOC)}$$

**Figure 1-6: Region-annotated language, RAL**



# Note:

- Dereferencing operations are implicit here.



# Example

For example, consider the following program to compute Fibonacci numbers[3]

```
fix fib. λn.
     if n<2 then 1
     else fib(n-2)+fib(n-1)
```

One possible region annotation of this program is (ignore everything but the first line for now)

```
fix fib. (λρi. (λρo. (λn.
     if new ρ. (n < (2 at ρ) then 1 at ρo)
     else new ρ1.
               new ρ2.fib[[ρ2]][[ρ1]] (new ρ.n -at ρ2 (2 at ρ))
          +at ρo new ρ3.fib[[ρ3]][[ρ1]] (new ρ.n -at ρ3 (1 at ρ))
     ) at ρi) at ρi) at ρf
```

# Completions for RAL.

$$
\begin{array}{rcl}
\|\mathtt{bv}\| & = & \mathtt{bv} \\
\left\|\begin{array}{l}\mathtt{if}\ t_0\ \mathtt{then}\ t_1 \\ \mathtt{else}\ t_2\end{array}\right\| & = & \begin{array}{l}\mathtt{if}\ \|t_0\|\ \mathtt{then}\ \|t_1\| \\ \mathtt{else}\ \|t_2\|\end{array} \\
\|x\| & = & x \\
\|(\lambda x.t)\ \mathtt{at}\ p\| & = & \lambda x.\|t\| \\
\left\|\langle \lambda x.t\rangle_p\right\| & = & \lambda x.\|t\|
\end{array}
\qquad
\begin{array}{rcl}
\|t_1\ t_2\| & = & \|t_1\|\ \|t_2\| \\
\|\mathtt{fix}\ x.u\| & = & \mathtt{fix}\ x.\|u\| \\
\|\mathtt{new}\ \rho.t\| & = & \|t\| \\
\|(\lambda \rho.u)\ \mathtt{at}\ p\| & = & \|u\| \\
\left\|\langle \lambda \rho.u\rangle_p\right\| & = & \|u\| \\
\|t\ [\![p]\!]\| & = & \|t\|
\end{array}
$$

**Figure 1-7: Definition of the erasure function**

*Type expressions*

$$
\begin{array}{rcll}
\mathtt{p} & \in & \text{Place} & \textit{places} \\
\epsilon & \in & \text{EffVar} & \textit{effect variables} \\
\varphi & \in & \mathcal{P}_{\text{fin}}(\text{Place} \cup \text{EffVar}) & \textit{effects} \\
\mathtt{T} & ::= & & \textit{type expressions:} \\
 & & \mathtt{X} & \textit{type variable} \\
 & & \mathtt{bool} & \textit{Boolean type} \\
 & & (\mathtt{T} \to {}^{\varphi}\mathtt{T}, \mathtt{p}) & \textit{function type} \\
 & & (\Pi\rho.{}^{\varphi}\mathtt{T}, \mathtt{p}) & \textit{region func.} \\
 & & \forall \mathtt{X}.\mathtt{T} & \textit{type polymorphism} \\
 & & \forall \epsilon.\mathtt{T} & \textit{effect polymorphism}
\end{array}
$$

*Typing rules* $\boxed{\Gamma \vdash \mathtt{t} :^{\varphi} \mathtt{T}}$

$$\frac{\Gamma(\mathtt{x}) = \mathtt{T}}{\Gamma \vdash \mathtt{x} :^{\varphi} \mathtt{T}} \quad \text{(TT-VAR)}$$

$$\Gamma \vdash \mathtt{bv} :^{\varphi} \mathtt{bool} \quad \text{(TT-BOOL)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 :^{\varphi} \mathtt{bool} \qquad \Gamma \vdash \mathtt{t}_2 :^{\varphi} \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3 :^{\varphi} \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 :^{\varphi} \mathtt{T}} \quad \text{(TT-IF)}$$

$$\frac{\Gamma, \mathtt{x} : \mathtt{T}_1 \vdash \mathtt{t} :^{\varphi_2} \mathtt{T}_2 \qquad \mathtt{p} \in \varphi}{\Gamma \vdash (\lambda \mathtt{x}.\mathtt{t})\ \mathtt{at}\ \mathtt{p} :^{\varphi} (\mathtt{T}_1 \to {}^{\varphi_2}\mathtt{T}_2, \mathtt{p})} \quad \text{(TT-ABS)}$$

$$\frac{\Gamma, \mathtt{x} : \mathtt{T}_1 \vdash \mathtt{t} :^{\varphi_2} \mathtt{T}_2}{\Gamma \vdash \langle \lambda \mathtt{x}.\mathtt{t} \rangle_{\mathtt{p}} :^{\varphi} (\mathtt{T}_1 \to {}^{\varphi_2}\mathtt{T}_2, \mathtt{p})} \quad \text{(TT-CLOS)}$$

$$\frac{\Gamma \vdash \mathtt{t}_0 :^{\varphi} (\mathtt{T}_1 \to {}^{\varphi_2}\mathtt{T}_2, \mathtt{p}) \qquad \Gamma \vdash \mathtt{t}_1 :^{\varphi} \mathtt{T}_1 \qquad \mathtt{p} \in \varphi \qquad \varphi_2 \subseteq \varphi}{\Gamma \vdash \mathtt{t}_0\ \mathtt{t}_1 :^{\varphi} \mathtt{T}_2} \quad \text{(TT-APP)}$$

$$\frac{\Gamma, \mathtt{x} : \mathtt{T} \vdash \mathtt{u} :^{\varphi} \mathtt{T}}{\Gamma \vdash \mathtt{fix}\ \mathtt{x}.\mathtt{u} :^{\varphi} \mathtt{T}} \quad \text{(TT-FIX)}$$

$$\frac{\Gamma \vdash \mathtt{u} :^{\varphi'} \mathtt{T} \qquad \rho \notin \mathrm{frv}(\Gamma) \qquad \mathtt{p} \in \varphi}{\Gamma \vdash (\lambda\rho.\mathtt{u})\ \mathtt{at}\ \mathtt{p} :^{\varphi} (\Pi\rho.{}^{\varphi'}\mathtt{T}, \mathtt{p})} \quad \text{(TT-RABS)}$$

$$\frac{\Gamma \vdash \mathtt{u} :^{\varphi'} \mathtt{T} \qquad \rho \notin \mathrm{frv}(\Gamma)}{\Gamma \vdash \langle \lambda\rho.\mathtt{u} \rangle_{\mathtt{p}} :^{\varphi} (\Pi\rho.{}^{\varphi'}\mathtt{T}, \mathtt{p})} \quad \text{(TT-RCLOS)}$$

$$\frac{\Gamma \vdash \mathtt{t} :^{\varphi} (\Pi\rho.{}^{\varphi'}\mathtt{T}, \mathtt{p}) \qquad \mathtt{p} \in \varphi \qquad [\rho \mapsto \mathtt{p}']\varphi' \subseteq \varphi}{\Gamma \vdash \mathtt{t}\ [\![\mathtt{p}']\!] :^{\varphi} [\rho \mapsto \mathtt{p}']\mathtt{T}} \quad \text{(TT-RAPP)}$$

$$\frac{\Gamma \vdash \mathtt{t} :^{\varphi, \rho} \mathtt{T} \qquad \rho \notin \mathrm{frv}(\Gamma, \mathtt{T})}{\Gamma \vdash \mathtt{new}\ \rho.\mathtt{t} :^{\varphi} \mathtt{T}} \quad \text{(TT-LETREG)}$$

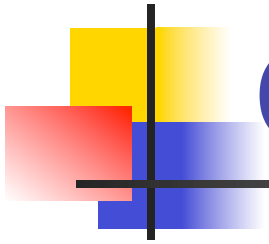
$$\frac{\Gamma \vdash \mathtt{t} :^{\varphi} \mathtt{T} \qquad \mathtt{X} \notin \mathrm{ftv}(\Gamma)}{\Gamma \vdash \mathtt{t} :^{\varphi} \forall \mathtt{X}.\mathtt{T}} \quad \text{(TT-TGEN)}$$

$$\frac{\Gamma \vdash \mathtt{t} :^{\varphi} \forall \mathtt{X}.\mathtt{T}}{\Gamma \vdash \mathtt{t} :^{\varphi} [\mathtt{X} \mapsto \mathtt{T}']\mathtt{T}} \quad \text{(TT-TINST)}$$

$$\frac{\Gamma \vdash \mathtt{t} :^{\varphi} \mathtt{T} \qquad \epsilon \notin \mathrm{fev}(\Gamma, \varphi)}{\Gamma \vdash \mathtt{t} :^{\varphi} \forall \epsilon.\mathtt{T}} \quad \text{(TT-EGEN)}$$

$$\frac{\Gamma \vdash \mathtt{t} :^{\varphi} \forall \epsilon.\mathtt{T}}{\Gamma \vdash \mathtt{t} :^{\varphi} [\epsilon \mapsto \varphi']\mathtt{T}} \quad \text{(TT-EINST)}$$

**Figure 1-8: The RTL region type system**



# Correctness

- RAL is partially correct.
- RTL (variant of Tofte-Talpin region type system) is sound.
- Consequently, RTL is correct.
- Follows from simple syntactic techniques (as illustrated before)

# The need for effect polymorphism

Consider the function *map.* Its region type is

$$\forall \alpha, \beta.(\alpha \rightarrow^{\varphi} \beta) \times (\alpha\ \mathtt{list}, \rho) \rightarrow^{\{\rho,\rho'\}\cup\varphi} (\beta\ \mathtt{list}, \rho')$$

Note that the effect and region variables must be universally quantified to allow map application in multiple contexts with cross contamination of region and effect information.

*New syntactic forms*

$$
\begin{array}{lcll}
\mathtt{t} & ::= & \ldots & \textit{terms:} \\
 & & (\mathtt{t} :: \mathtt{t})\ \mathtt{at}\ \mathtt{p} & \textit{list constructor} \\
 & & \mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \begin{smallmatrix}\mathtt{nil} \Rightarrow \mathtt{t}_1 \\ (\mathtt{x} :: \mathtt{x}') \Rightarrow \mathtt{t}_2\end{smallmatrix} & \textit{case on lists} \\
\mathtt{v} & ::= & \ldots & \textit{values:} \\
 & & \langle \mathtt{v} :: \mathtt{v} \rangle_{\mathtt{p}} & \textit{cons cell} \\
 & & \mathtt{nil} & \textit{empty list} \\
\mathtt{T} & ::= & \ldots & \textit{types:} \\
 & & (\mathtt{T}\ \mathtt{list}, \mathtt{p}) & \textit{type of lists}
\end{array}
$$

*New erasure rules*

$$
\begin{array}{rcl}
\|\mathtt{nil}\| & = & \mathtt{nil} \\
\|(\mathtt{t}_1 :: \mathtt{t}_2)\ \mathtt{at}\ \mathtt{p}\| & = & \|\mathtt{t}_1\| :: \|\mathtt{t}_2\| \\
\left\|\langle \mathtt{t}_1 :: \mathtt{t}_2 \rangle_{\mathtt{p}}\right\| & = & \|\mathtt{t}_1\| :: \|\mathtt{t}_2\| \\
\left\|\mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \begin{smallmatrix}\mathtt{nil} \Rightarrow \mathtt{t}_1 \\ (\mathtt{x} :: \mathtt{x}') \Rightarrow \mathtt{t}_2\end{smallmatrix}\right\| & = & \\
\multicolumn{3}{r}{\mathtt{case}\ \|\mathtt{t}_0\|\ \mathtt{of}\ \begin{smallmatrix}\mathtt{nil} \Rightarrow \|\mathtt{t}_1\| \\ (\mathtt{x} :: \mathtt{x}') \Rightarrow \|\mathtt{t}_2\|\end{smallmatrix}}
\end{array}
$$

*New evaluation rules* $\boxed{\mathtt{t} \xrightarrow{\text{RAL}} \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \xrightarrow{\text{RAL}} \mathtt{t}_1'}{(\mathtt{t}_1 :: \mathtt{t}_2)\ \mathtt{at}\ \mathtt{p} \xrightarrow{\text{RAL}} (\mathtt{t}_1' :: \mathtt{t}_2)\ \mathtt{at}\ \mathtt{p}}\ \text{(E-Cons1)}$$

$$\frac{\mathtt{t}_2 \xrightarrow{\text{RAL}} \mathtt{t}_2'}{(\mathtt{v}_1 :: \mathtt{t}_2)\ \mathtt{at}\ \mathtt{p} \xrightarrow{\text{RAL}} (\mathtt{v}_1 :: \mathtt{t}_2')\ \mathtt{at}\ \mathtt{p}}\ \text{(E-Cons2)}$$

$$(\mathtt{v}_1 :: \mathtt{v}_2)\ \mathtt{at}\ \rho \xrightarrow{\text{RAL}} \langle \mathtt{v}_1 :: \mathtt{v}_2 \rangle_{\rho}\ \text{(E-ConsAlloc)}$$

$$\frac{\mathtt{t}_0 \xrightarrow{\text{RAL}} \mathtt{t}_0'}{\begin{array}{l}\mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \mathtt{nil} \Rightarrow \mathtt{t}_1 \mid (\mathtt{x} :: \mathtt{x}') \Rightarrow \mathtt{t}_2 \\ \longrightarrow \mathtt{case}\ \mathtt{t}_0'\ \mathtt{of}\ \mathtt{nil} \Rightarrow \mathtt{t}_1 \mid (\mathtt{x} :: \mathtt{x}') \Rightarrow \mathtt{t}_2\end{array}}\ \text{(E-Case)}$$

$$\mathtt{case}\ \mathtt{nil}\ \mathtt{of}\ \mathtt{nil} \Rightarrow \mathtt{t}_1 \mid (\mathtt{x} :: \mathtt{x}') \Rightarrow \mathtt{t}_2 \xrightarrow{\text{RAL}} \mathtt{t}_1\ \text{(E-CaseNil)}$$

$$\mathtt{case}\ \langle \mathtt{v} :: \mathtt{v}' \rangle_{\rho}\ \mathtt{of}\ \mathtt{nil} \Rightarrow \mathtt{t}_1 \mid (\mathtt{x} :: \mathtt{x}') \Rightarrow \mathtt{t}_2 \longrightarrow [\mathtt{x}' \mapsto \mathtt{v}'][\mathtt{x} \mapsto \mathtt{v}]\mathtt{t}_2\ \text{(E-CaseCons)}$$

*New typing rules* $\boxed{\Gamma \vdash \mathtt{t} : \varphi\ \mathtt{T}}$

$$\Gamma \vdash \mathtt{nil} : \varphi\ (\mathtt{T}\ \mathtt{list}, \mathtt{p})\ \text{(RT-Nil)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \varphi\ \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_2 : \varphi\ (\mathtt{T}\ \mathtt{list}, \mathtt{p}) \qquad \mathtt{p} \in \varphi}{\Gamma \vdash (\mathtt{t}_1 :: \mathtt{t}_2)\ \mathtt{at}\ \mathtt{p} : \varphi\ (\mathtt{T}\ \mathtt{list}, \mathtt{p})}\ \text{(RT-Cons)}$$

$$\frac{\Gamma \vdash \mathtt{v}_1 : \varphi\ \mathtt{T} \qquad \Gamma \vdash \mathtt{v}_2 : \varphi\ (\mathtt{T}\ \mathtt{list}, \mathtt{p})}{\Gamma \vdash \langle \mathtt{v}_1 :: \mathtt{v}_2 \rangle_{\mathtt{p}} : \varphi\ (\mathtt{T}\ \mathtt{list}, \mathtt{p})}\ \text{(RT-ConsCell)}$$

$$\frac{\begin{array}{ll}\Gamma \vdash \mathtt{t}_0 : \varphi\ \mathtt{T}' & \mathtt{T}' = (\mathtt{T}\ \mathtt{list}, \mathtt{p}) \qquad \mathtt{p} \in \varphi \\ \Gamma \vdash \mathtt{t}_1 : \varphi\ \mathtt{T}'' & \Gamma, \mathtt{x} : \mathtt{T}, \mathtt{x}' : \mathtt{T}' \vdash \mathtt{t}_2 : \varphi\ \mathtt{T}''\end{array}}{\Gamma \vdash \mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \begin{smallmatrix}\mathtt{nil} \Rightarrow \mathtt{t}_1 \\ (\mathtt{x} :: \mathtt{x}') \Rightarrow \mathtt{t}_2\end{smallmatrix} : \varphi\ \mathtt{T}''}\ \text{(RT-Case)}$$

**Figure 1-9: Extending the system with a list type**

# Game of Life: Tofte/Talpin completion

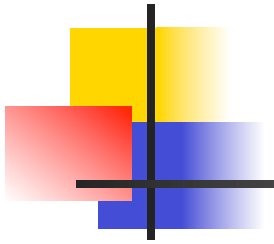
```
let rec nextgen(g) = ⟨read g; create and return new generation⟩
let rec life(n,g) = if n=0 then g
                    else life(n-1,nextgen(g))
```

```
letrec nextgen[ρ](g) = ⟨read g from ρ; create new gen. at ρ⟩
letrec life[ρn, ρg](n,g) =
       if n=0 then g
       else new ρ'n
            in life[ρ'n, ρg]((n-1) at ρ'n, nextgen[ρg](g))
```

Note:

- life not tail recursive!
- result of nextgen must be in same region as argument

# Region inference (Tofte/ Birkedal 2001)



- Inference for expressions:
  - Build derivation template with uniquely occurring region and effect (meta)variables
  - Collect equational constraints between region variables and containment contraints between effects variables
  - Normalize (eliminate) constraints by producing substitution and making scoped region decisions
- Types of fixpoints (recursive functions):
  - Employ Kleene-Mycroft iteration: Assume "most polymorphic" type for recursive occurrences of function, compute type according to process above.
  - Does that terminate?

# Region inference: Example

```
letrec m(f) = if f(0) then 0 else m(λx.f(x+1)) + 1
in m(λx.x=10)
```

Assume that the recursive occurrence of m has type

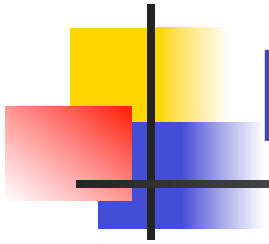
$$\forall \epsilon_1, \epsilon_2.\Pi\rho_1, \rho_2.^{\{\rho_2\}}((\texttt{int} \rightarrow^{\{\epsilon_1\}} \texttt{bool}, \rho_1) \rightarrow^{\{\epsilon_2,\epsilon_1,\rho_1\}} \texttt{int}, \rho_2).$$

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \texttt{m}\,[\![\rho_4, \rho_5]\!] :^{\varphi} ((\texttt{int} \rightarrow^{\varphi_4} \texttt{bool}, \rho_4) \rightarrow^{\varphi_5} \texttt{int}, \rho_5)} \quad \dfrac{\dfrac{\dfrac{}{\Gamma' \vdash \texttt{f} :^{\varphi_1} (\texttt{int} \rightarrow^{\varphi_3} \texttt{bool}, \rho_3)} \quad \dfrac{\dfrac{}{\Gamma' \vdash \texttt{x} :^{\varphi_4} \texttt{int}}}{\Gamma' \vdash \texttt{x+1} :^{\varphi_4} \texttt{int}}}{\Gamma' \vdash \texttt{f}\,(\texttt{x+1}) :^{\varphi_4} \texttt{bool}}}{\Gamma \vdash (\lambda\texttt{x.f}\,(\texttt{x+1}))\ \texttt{at}\ \rho_4 :^{\varphi} (\texttt{int} \rightarrow^{\varphi_4} \texttt{bool}, \rho_4)}}{\Gamma \vdash \texttt{m}\,[\![\rho_4, \rho_5]\!]\,((\lambda\texttt{x.f}\,(\texttt{x+1}))\ \texttt{at}\ \rho_4) :^{\varphi} \texttt{int}}$$

where $\Gamma$ is $\texttt{m} : \forall \epsilon_1, \epsilon_2.\Pi\rho_1, \rho_2.^{\{\rho_2\}}((\texttt{int} \rightarrow^{\{\epsilon_1\}} \texttt{bool}, \rho_1) \rightarrow^{\{\epsilon_2,\epsilon_1,\rho_1\}} \texttt{int}, \rho_2), \texttt{f} : (\texttt{int} \rightarrow^{\varphi_3} \texttt{bool}, \rho_3)$ and $\Gamma'$ is $\Gamma, \texttt{x} : \texttt{int}$.

Collected effect constraints: $\varphi_4 \subseteq \varphi_5, \rho_4 \in \varphi_5, \{\rho_5\} \subseteq \varphi, \varphi_3 \subseteq \varphi_4, \rho_3 \in \varphi_4, \rho_4 \in \varphi, \varphi_5 \subseteq \varphi, \rho_5 \in \varphi$.

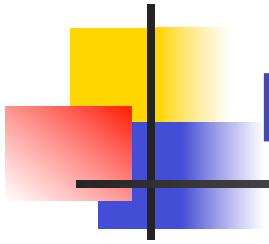
**Figure 1-10: A partially region-inferred proof tree.**



# Musings

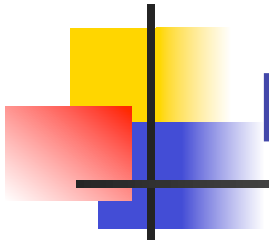
- Tofte/Birkedal's inference is not known to produce a principal completion for input programs (under suitable definition of subsumption for polymorphic types).
- Tofte/Talpin's region system (of which RTL is a variant) seeks to be 'purely' equational, but *does* generate effect subtype constraints.
- Yet it does not have any subtype qualifications in its polymorphic types (which are usually required when combining parametric polymorphism and subtyping for principality)

# Extensions to Tofte/Talpin region inference

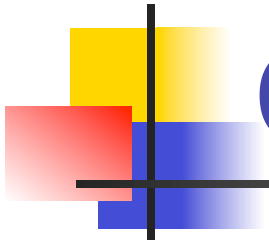


- Birkedal-Tofte-Vejlstrup: Region resetting in ML Kit
- Aiken-Faehndrich-Levin: Reducing lag and drag by moving allocation and deallocation sites closer to first, resp. last region access
- Walker-Crary-Morrisett, Walker-Watkins, Henglein-Makholm-Niss: Typing calculi for decoupled allocation and deallocation operations

# Extensions to Tofte/Talpin region inference



- Cyclone (Grossman et al., Fluet-Morrisett): region lifetime subtyping
- Vault (Faehndrich-DeLine): safe transitions between linear and nonlinear usage
- Boyapati-Salcianu-Beebee-Rinard: Ownership types and regions
- Christiansen-Velschow, Chin-Craciun-Qin-Rinard: region inference for OO languages
- Makholm-Sagonas: region inference for logic programs



# Contributions

- Region-based memory management as nonstandard interpretation of type-based value flow analysis
- Correctness = Partial correctness + soundness
  - Partial correctness (by partial simulation): depends only instrumented semantics (w/o type system)
  - Soundness (by progress and preserveation): depends only on type system
- Very simple proofs!

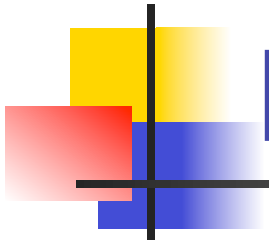
# Type-based analysis: "The method"

- Instrumented typed semantics:
  - additional constructs (annotations), with
  - operational semantics and
  - type system for annotated language
- TBA specifies *solution space (completions)*: Set of all possible derivations for given source program
  - Correctness: partial correctness + soundness (or: proof by coherence of completions, e.g. Henglein, "Dynamic Typing: Syntax and Proof Theory", SCP, 1994)
  - Quality: Choice between completions, from trivial to principal, depending on context (e.g., based on a subsumption theory for completions)
- Analysis = transformation: Source program → well-typed annotated program. (The well-typed annotated program *is* the result of the analysis.)
- Exploitation of analysis = (possibly nonstandard) implementation or further processing of annotated program

# Game of Life (Postscript): ML Kit with Regions

```
let rec copy(g) = ⟨read g; make fresh copy⟩
let rec life'((n,g) as p)
        = if n=0 then p
          else life'(n-1,copy(nextgen(g)))
let rec life(p) = snd (life' (p))
```

```
letrec nextgen[ρ, ρ'](g) = ⟨read g from ρ; new gen. at ρ'⟩
letrec copy[ρ', ρ](g) = ⟨read g from ρ'; fresh copy atbot ρ⟩
letrec life'[ρn, ρg]((n,g) as p)
       = if n=0 then p
         else life'[ρn, ρg]((n-1) atbot ρn,
                            new ρg'
                                                in copy[ρg', atbot ρg]
                                                   (nextgen[ρg, ρg'](g)))
letrec life[ρn, ρg](p) = snd (life'[ρn, ρg](p))
```



# Exercises

- Read "Effect types and region-based memory management", 3-3.3 (plus the rest at your leisure). Skip the "Notes" sections the first time around.
- Do the following exercises: 3.2.4-3.2.6, 3.3.2, 3.4.2, (3.2.11)
- STL is unsound. Is there a way of *restricting* the language instead of adding effects to ensure soundness?