Program repair as sound optimization of broken programs

Tarmo Uustalu, IoC

joint work with Ando Saabas, Skype,
Bernd Fischer, U. of Southampton

TSEM, 12 November 2009

Program repair: the dream

- Program repair: fixing a broken program (a program that may abort), by transforming it into a safe or safer program (one that cannot evaluate abnormally or will do so less often).
- The transformation should be *compile-time* and *automatic* (although subject to review by the programmer).
- It should also be defendable, e.g., as embodying a plausible method of reconstructing programmer intent.
- Mathematically, it should be *sound* for a suitable notion of validity.

Program repair: our approach

- Two central ideas:
  - fix the meaning of broken programs by a dedicated *error-compensating* semantics,
    $\leadsto$ the psychological issue of programmer intent is isolated into the definition of this semantics,
  - guide transformation by a program analysis, with analysis results interpreted *relationally*,
    $\leadsto$ program repair becomes similar to sound program optimization
- In fact we get a spectrum:
  – program repair
    – enforcement of coding conventions
      – program optimization.

Error-compensating semantics

- To fix the intended meaning of broken programs (programs that may abort under the standard *error-admitting* semantics), we assign the programming language a special *error-compensating* semantics with no or fewer abnormal evaluations.
- On safe programs, the two semantics must agree.
- The evaluations of a given program under the error-compensating semantics should agree with those of the repaired program under the error-admitting semantics.
- If the given program is already safe, the repair may only optimize it.

Relationally interpreted types

- Our repairs are based on program analyses, described as type systems.
- The types are interpreted as *relations* between states of the error-compensating and error-admitting semantics.
- Validity of repair, i.e., the agreement between the evaluations of the given and repaired program is defined in terms of these relations.

Example: Repairing file access errors

- Error-admitting semantics: Opening an open file, reading or closing a closed file cause abortion.
- Error-compensating semantics: Opening, reading, closing are always possible. In essence, all files are always open. Opening and closing reset the file pointer.
- Rationale behind: Likely, the programmer may have forgotten some opens and closes.
- Repair:
    - *removes all* closes and opens,
    - *inserts some*, generally elsewhere, to render all reads safe and belonging appropriately to the same or different sessions, minimizing session lengths.

(Cf. partial redundancy elimination: expression evaluations removed and reinserted.)

- E.g.,

$\text{read}(f, x);$ $\text{read}(f, y);$ $\text{open}(f);$ $\text{read}(f, z);$ $w := x - z;$ $\text{close}(f)$	$\hookrightarrow$	$\text{open}(f); \text{read}(f, x);$ $\text{read}(f, y); \text{close}(f);$ $\text{open}(f); \text{read}(f, z); \text{close}(f);$ $w := x - z$

$\text{if } b \text{ then}$ $\quad \text{read}(f, x)$ else $\quad x := x + 1;$ $\text{read}(f, y)$	$\hookrightarrow$	$\text{if } b \text{ then}$ $\quad \text{open}(f); \text{read}(f, x)$ else $\quad x := x + 1;$ $\quad \text{open}(f);$ $\text{read}(f, y); \text{close}(f)$

Error-admitting semantics

States: $\sigma \in \textbf{Var} \longrightarrow \mathbb{Z}$, $\rho \in \textbf{F} \longrightarrow \{\mathsf{c}\} + \{\mathsf{o}(n) \mid n \in \mathbb{N}\}$
(closed or open and at some line)

Evaluation rules:

$$\frac{\rho(f) = \mathsf{c}}{\sigma, \rho \succ\!\mathsf{open}(f)\!\rightarrow\ \sigma, \rho[f \mapsto \mathsf{o}(0)]} \qquad \frac{\rho(f) = \mathsf{o}(n)}{\sigma, \rho \succ\!\mathsf{close}(f)\!\rightarrow\ \sigma, \rho[f \mapsto \mathsf{c}]}$$

$$\frac{\rho(f) = \mathsf{o}(n)}{\sigma, \rho \succ\!\mathsf{read}(f, x)\!\rightarrow\ \sigma[x \mapsto \phi(f, n)], \rho[f \mapsto \mathsf{o}(n+1)]}$$

$$\frac{\rho(f) = \mathsf{o}(n)}{\sigma, \rho \succ\!\mathsf{open}(f)\!\rightarrow\!\!\mid} \qquad \frac{\rho(f) = \mathsf{c}}{\sigma, \rho \succ\!\mathsf{close}(f)\!\rightarrow\!\!\mid} \qquad \frac{\rho(f) = \mathsf{c}}{\sigma, \rho \succ\!\mathsf{read}(f, x)\!\rightarrow\!\!\mid}$$

Safety type system

Types: $d \in \mathbf{F} \to \{\mathsf{c}, \mathsf{o}\}$ (closed, open)

Typing rules:

$$\frac{d(f) = \mathsf{c}}{\mathsf{open}(f) : d \longrightarrow d[f \mapsto \mathsf{o}]} \qquad \frac{d(f) = \mathsf{o}}{\mathsf{close}(f) : d \longrightarrow d[f \mapsto \mathsf{c}]}$$

$$\frac{d(f) = \mathsf{o}}{\mathsf{read}(f, x) : d \longrightarrow d}$$

no subsumption rule

Types as predicates on states:

$$\frac{}{\mathsf{o}(n) \models \mathsf{o}} \qquad \frac{}{\mathsf{c} \models \mathsf{c}} \qquad \frac{\forall f \in \mathbf{F}.\, \rho(f) \models d(f)}{(\sigma, \rho) \models d}$$

Soundness of the safety type system

If $s : d \longrightarrow d'$ in the safety type system, then

1. if $(\sigma, \rho) \models d$ and $(\sigma, \rho) \succ s \rightarrow (\sigma', \rho')$ in the error-admitting semantics, then $(\sigma', \rho') \models d'$,
2. it cannot be that $(\sigma, \rho) \models d$ and $(\sigma, \rho) \succ s \rightarrowtail$ in the error-admitting semantics.

Error-compensating semantics

States: $\sigma \in \mathbf{Var} \longrightarrow \mathbb{Z}$, $\rho : \mathbf{F} \longrightarrow \mathbb{N}$.

Evaluation rules:

$$\frac{}{\sigma, \rho \succ \mathsf{open}(f) \rightarrow \sigma, \rho[f \mapsto 0]} \qquad \frac{}{\sigma, \rho \succ \mathsf{close}(f) \rightarrow \sigma, \rho[f \mapsto 0]}$$

$$\frac{}{\sigma, \rho \succ \mathsf{read}(f, x) \rightarrow \sigma[x \mapsto \phi(f, \rho(f))], \rho[f \mapsto \rho(f) + 1]}$$

(no abnormal evaluations)

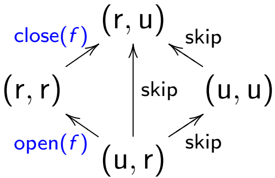
Repair type system

Types: $d, e \in \mathbf{F} \to \{\mathrm{r}, \mathrm{u}\}$ (possibly read/certainly unread before, after)

Subtyping:

$$\frac{}{(\mathrm{u}, \mathrm{r}) \leq (\mathrm{r}, \mathrm{r}) \hookrightarrow_f \mathsf{open}(f)} \qquad \frac{}{(\mathrm{r}, \mathrm{r}) \leq (\mathrm{r}, \mathrm{u}) \hookrightarrow_f \mathsf{close}(f)}$$

$$\frac{}{(m, m) \leq (m, m) \hookrightarrow_f \mathsf{skip}} \qquad \frac{}{(\mathrm{u}, m) \leq (m', \mathrm{u}) \hookrightarrow_f \mathsf{skip}}$$



$$\frac{\forall f \in \mathbf{F}.\ (d(f), e(f)) \leq (d'(f), e'(f)) \hookrightarrow_f s(f)}{(d, e) \leq (d', e') \hookrightarrow [s(f) \mid f \in \mathbf{F}]}$$

Repair type system ctd.

Typing rules:

$$\frac{}{\textcolor{red}{\mathsf{open}(f)} : (d, e[f \mapsto \mathsf{u}]) \longrightarrow (d[f \mapsto \mathsf{u}], e) \hookrightarrow \mathsf{skip}}$$

$$\frac{}{\textcolor{red}{\mathsf{close}(f)} : (d, e[f \mapsto \mathsf{u}]) \longrightarrow (d[f \mapsto \mathsf{u}], e) \hookrightarrow \mathsf{skip}}$$

$$\frac{d(f) = e(f) = \mathsf{r}}{\mathsf{read}(f, x) : (d, e) \longrightarrow (d, e) \hookrightarrow \mathsf{read}(f, x)}$$

$$\frac{(d, e) \leq (d_0, e_0) \hookrightarrow s_{pre} \qquad s : (d_0, e_0) \longrightarrow (d_0', e_0') \hookrightarrow s_* \qquad (d_0', e_0') \leq (d', e') \hookrightarrow s_{post}}{s : (d, e) \longrightarrow (d', e') \hookrightarrow s_{pre}; s_*; s_{post}}$$

Types as relations:

$$\frac{}{n \sim_{(\mathsf{r},\mathsf{r})} \mathsf{o}(n)} \qquad \frac{}{0 \sim_{(\mathsf{u},\mathsf{r})} \mathsf{c}} \qquad \frac{}{n \sim_{(m,\mathsf{u})} \mathsf{c}} \qquad \frac{\forall f \in \mathbf{F}.\ \rho(f) \sim_{(d,e)} \rho_*(f)}{(\sigma, \rho) \sim_{(d,e)} (\sigma, \rho_*)}$$

Soundness of the repair type system

If $s : (d, e) \longrightarrow (d', e') \hookrightarrow s_*$ in the repair type system, then

1. if $(\sigma, \rho) \sim_{(d,e)} (\sigma_*, \rho_*)$ and $(\sigma, \rho) \succ s \rightarrow (\sigma', \rho')$ in the error-compensating semantics, then there exists (σ'_*, ρ'_*) such that $(\sigma', \rho') \sim_{(d',e')} (\sigma'_*, \rho'_*)$ and $(\sigma_*, \rho_*) \succ s_* \rightarrow (\sigma'_*, \rho'_*)$ in the error-admitting semantics;
2. if $(\sigma, \rho) \sim_{(d,e)} (\sigma_*, \rho_*)$ and $(\sigma_*, \rho_*) \succ s_* \rightarrow (\sigma'_*, \rho'_*)$ in the error-admitting semantics, then there exists (σ', ρ') such that $(\sigma', \rho') \sim_{(d',e')} (\sigma'_*, \rho'_*)$ and $(\sigma, \rho) \succ s \rightarrow (\sigma', \rho')$ in the error-compensating semantics;
3. it cannot be that $(\sigma, \rho) \sim_{(d,e)} (\sigma_*, \rho_*)$ and $(\sigma_*, \rho_*) \succ s_* \rightarrow\!\!|$ in the error-admitting semantics;
4. $s_* : (d, e)^\# \longrightarrow (d', e')^\#$ where $(\mathrm{r}, \mathrm{r})^\# =_{\mathrm{df}} \mathrm{o}$, $(m, \mathrm{u})^\# =_{\mathrm{df}} \mathrm{c}$, $(\mathrm{u}, m)^\# =_{\mathrm{df}} \mathrm{c}$ and $(d, e)^\#(f) =_{\mathrm{df}} (d(f), e(f))^\#$.

Example: Queue access

- Error-admitting semantics: overflow, underflow lead to abortion.
- Error-compensating semantics: some platform-specific implementation (e.g., enqueues to a full queue skipped, dequeues from an empty queue return some default value)
- Rationale: Compensation given by an implementation.
- Program repair: based on an interval analysis about queue length, makes it explicit what the compensation does.

Error-admitting semantics

States: $\sigma \in \mathbf{Var} \longrightarrow \mathbb{Z}$, $q \in \mathbb{Z}^*$, $|q| \leq N$ for a fixed $N \in \mathbb{N}$

Evaluation rules:

$$\frac{|q| < N}{\sigma, q \succ\!\mathsf{enq}(a)\!\rightarrow\ \sigma, q +\!\!+ [[\![a]\!]\sigma]} \qquad \frac{}{\sigma, v : q \succ\!\mathsf{deq}(x)\!\rightarrow\ \sigma[x \mapsto v], q}$$

$$\frac{|q| = N}{\sigma, q \succ\!\mathsf{enq}(a)\!\rightarrow\!\!\!\mid} \qquad \frac{}{\sigma, [] \succ\!\mathsf{deq}(x)\!\rightarrow\!\!\!\mid}$$

Safety type system

Types: $lo, hi \in \mathbb{N}$, $lo \leq hi$

Subtyping rules:

$$\frac{lo' \leq lo \quad hi \leq hi'}{[lo, hi] \leq [lo', hi']}$$

Typing rules:

$$\frac{hi < N}{\mathrm{enq}(a) : [lo, hi] \longrightarrow [lo+1, hi+1]} \qquad \frac{0 < lo}{\mathrm{deq}(x) : [lo, hi] \longrightarrow [lo-1, hi-1]}$$

$$\frac{[lo, hi] \leq [lo_0, hi_0] \quad s : [lo_0, hi_0] \longrightarrow [lo_0', hi_0'] \quad [lo_0', hi_0'] \leq [lo', hi']}{s : [lo, hi] \longrightarrow [lo', hi']}$$

Error-compensating semantics

States as in the error-admitting semantics

Evaluation rules:

$$\frac{|q| < N}{\sigma, q \succ\!\text{enq}(a)\!\rightarrow \sigma, q +\!\!+ [[\![a]\!]\sigma]} \qquad \frac{|q| = N}{\sigma, q \succ\!\text{enq}(a)\!\rightarrow \sigma, q}$$

$$\frac{}{\sigma, v : q \succ\!\text{deq}(x)\!\rightarrow \sigma[x \mapsto v], q} \qquad \frac{}{\sigma, [] \succ\!\text{deq}(x)\!\rightarrow \sigma[x \mapsto 0], []}$$

Repair type system

Types as in the safety type system

Subtyping rules:

$$\frac{lo' \leq lo \quad hi \leq hi'}{[lo, hi] \leq [lo', hi']}$$

Typing rules:

$$\frac{hi < N}{\mathsf{enq}(a) : [lo, hi] \longrightarrow [lo+1, hi+1] \hookrightarrow \mathsf{enq}(a)}$$

$$\frac{}{\mathsf{enq}(a) : [N, N] \longrightarrow [N, N] \hookrightarrow \mathsf{skip}}$$

$$\frac{lo < N}{\mathsf{enq}(a) : [lo, N] \longrightarrow [lo+1, N] \hookrightarrow \mathsf{if}\ \neg\mathsf{full}\ \mathsf{then}\ \mathsf{enq}(a)\ \mathsf{else}\ \mathsf{skip}}$$

$$\frac{0 < lo}{\mathsf{deq}(x) : [lo, hi] \longrightarrow [lo - 1, hi - 1] \hookrightarrow \mathsf{deq}(x)} \qquad \frac{}{\mathsf{deq}(x) : [0, 0] \longrightarrow [0, 0] \hookrightarrow x := 0}$$

$$\frac{0 < hi}{\mathsf{deq}(x) : [0, hi] \longrightarrow [0, hi - 1] \hookrightarrow \mathsf{if}\ \neg \mathsf{emp}\ \mathsf{then}\ \mathsf{deq}(x)\ \mathsf{else}\ x := 0}$$

$$\frac{[lo_0, hi_0] \leq [lo, hi] \quad s : [lo, hi] \longrightarrow [lo', hi'] \hookrightarrow s_* \quad [lo', hi'] \leq [lo'_0, hi'_0]}{s : [lo_0, hi_0] \longrightarrow [lo'_0, hi'_0] \hookrightarrow s_*}$$

Example: Modular arithmetic

- Error-admitting semantics: ideal arithmetic (in $[0..N-1]$).
- Error-compensating semantics: arithmetic modulo N.
- Program repair: based on an interval analysis about values of variables, inserts explicit mods (but not more than indispensable).
- Transformation of a proof about the repaired program to a proof about a given program makes it possible to reason in the ideal arithmetic and transfer the argument to modular arithmetic (with proof transformation inserting the interval reasoning).

Conclusion

- Program repair can be put on a firm semantic footing. The psychological engineering issue of reconstructing programmer intent can be isolated.
- The challenge is, given an error-compensating semantics, to find a suitable program analysis with a suitable semantical interpretation.
- This set up, the type-systematic method makes soundness proofs relatively straightforward checks also leading to automatic transformations of program correctness proofs.