Error Handling

Giuseppe Primiero

FWO - Research Foundation Flanders
Centre for Logic and Philosophy of Science, Ghent University

Giuseppe.Primiero@Ugent.be
http://www.philosophy.ugent.be/giuseppeprimiero/

GROLOG Lecture Series - 21 March 2013 - Groningen

Intro

- Psychology: very large literature on practical errors, see e.g. [Reason, 1990], [Woods, 2010], [Dekker, 2011];
- Epistemology&Philosophy of Science: error detection and resolution in paradigm definition and change (Popper, Lakatos, Kuhn, Bayesian epistemology); see e.g. [Mayo, 1996], [Allchin, 2001], [Mayo, 2010];
- Applications: fault tolerance ([Goodenough, 1975], [Cristian, 1982]); specification design; technological malfunctioning; see e.g. [Turner, 2011].

Logic

- Philosophical Logic: defeasible conditions and bounded resources for knowledge as approximations to errors; see e.g. [Williamson, 1992]; [Williamson, 2002]; [Woods, 2004]; [Sundholm, 2012]; [Bonnay and Egre', 2011];

- What about formal logic?: PDEL; proof-theoretical approach.

Tasks

1. formulate a full, empirically-informed characterization of error states for informational systems; presented in [Primiero, 2013];
2. TODAY: a formal model of logical processes with error states.

Outline

Types

- Types are semantic categories of objects:
  - $n = 0$ and $S(n)$ are terms of the type *Nat*;
  - *true*; *false* are terms of type *Bool*;
  - values on a continuous line are terms of type *Real*;
  - *proofs* are terms of the type *Prop*;
  - *programs* are terms of the type *Spec*;

Types

- Types are semantic categories of objects:
  - $n = 0$ and $S(n)$ are terms of the type *Nat*;
  - *true*; *false* are terms of type *Bool*;
  - values on a continuous line are terms of type *Real*;
  - *proofs* are terms of the type *Prop*;
  - *programs* are terms of the type *Spec*;
- Complex expressions are construed by functional operations:
  - introduction and elimination for connectives (proof-theoretical style)
  - abstraction and applications (λ-style)
  - predications over sets of functions for quantifiers (Σ, Π)

Types

- Types are semantic categories of objects:
  - $n = 0$ and $S(n)$ are terms of the type *Nat*;
  - *true*; *false* are terms of type *Bool*;
  - values on a continuous line are terms of type *Real*;
  - *proofs* are terms of the type *Prop*;
  - *programs* are terms of the type *Spec*;
- Complex expressions are construed by functional operations:
  - introduction and elimination for connectives (proof-theoretical style)
  - abstraction and applications (λ-style)
  - predications over sets of functions for quantifiers (Σ, Π)
- validity is by type-checking (truth for *Prop*; termination for *Spec*).

A Basic Language for the Type of Propositions

Definition (Syntax)

The basic alphabet is composed as follows:

$\texttt{Terms} := x \mid a$, variables and constants;

$\texttt{Types} := \alpha \mid \bot \mid \alpha + \beta \mid \alpha \times \beta \mid \alpha \to \beta$

Connectives

$$\frac{a : \alpha \qquad b : \beta}{(a, b) : \alpha \times \beta} I\times \qquad \frac{(a, b) : \alpha \times \beta}{a : \alpha} E \times (l)$$

$$\frac{a : \alpha}{a : \alpha + \beta} I + (1) \qquad \frac{b : \beta}{b : \alpha + \beta} I + (2)$$

$$\frac{\alpha + \beta \qquad a : \alpha \vdash c : \gamma \qquad b : \beta \vdash c : \gamma}{c : \gamma} E+$$

$$\frac{a : \alpha \qquad \alpha \vdash b : \beta}{(x)b : \alpha \to \beta} I \to \frac{(x)b : \alpha \to \beta \qquad (x/a) : \alpha}{\alpha \vdash b : \beta} E \to$$

Standard $\bot$

Standard rule for falsehood (a similar approach in PDEL):

$$\frac{\Delta, \Gamma \vdash m : \bot}{\Delta; \Gamma \vdash ABORT(m) : \alpha}$$

Standard $\bot$

Standard rule for falsehood (a similar approach in PDEL):

$$\frac{\Delta, \Gamma \vdash m : \bot}{\Delta; \Gamma \vdash ABORT(m) : \alpha}$$

1. No use of the distinction between local Γ and global resources Δ;
2. No internal structure for the error term m;
3. No available recovery from *ABORT*.

Wishlist

1. Control on local/global validity;
2. Define minimal (in-)correctness conditions;
3. Analyse errors in view of the different conditions breaches;
4. Design formal strategies to identify and correct errors.

A Premise: Errors as Limits of Validity

- Errors as emergence of uncertainty;
- To express uncertainty, we need to limit validity;
- Our calculus will admit *local* validity.

Step 1: Add Local Execution Functions

- *run*$_i$: execution of a term bounded to a location; call by value;
- *exec*: global unbounded value, valid for every other term to call; call by name;
- $\rightarrow$: to express implication from *exec*;
- $\supset$: to express implication from *run*;
- *synchro*: to pass a $\supset$ construction to an *exec* value.

Modified Syntax

Definition (Syntax)

Functions and their values are given as:

$\texttt{Terms} := x_i \mid a_i$, for $i \in \texttt{I}$

$\texttt{Types} := \alpha \mid \bot \mid \alpha \times \beta \mid \alpha + \beta \mid \alpha \to \beta \mid \alpha \supset \beta$

$\texttt{Locations} := 1 < \cdots < n$

$\texttt{Operations} := exec(\alpha) \mid run_i(\alpha) \mid run_{i \cup j}(\alpha \cdot \beta) \mid run_{i \cap j}(\alpha \cdot \beta) \mid synchro_j(\beta(exec(\alpha)))$, where $\cdot \in \{+, \times\}$

Connectives I

$$\frac{}{\Delta_i, a_i : \alpha \vdash exec(\alpha)} \text{ Global}$$

$$\frac{a_i : \alpha \vdash exec(\alpha) \qquad b_j : \beta \vdash exec(\beta)}{\vdash run_{i \cap j}(\alpha \times \beta)} \; I\times$$

$$\frac{\vdash run_{i \cap j}(\alpha \times \beta)}{\vdash exec(\alpha)} \; E \times (l)$$

$$\frac{a_i : \alpha \vdash exec(\alpha)}{\vdash run_i(\alpha + \beta)} \; I + (l) \qquad \frac{b_j : \beta \vdash exec(\beta)}{run_j(\alpha + \beta)} \; I + (r)$$

$$\frac{\vdash run_{i \cup j}(\alpha + \beta) \qquad run_i(\alpha) \vdash c_k : \gamma \qquad run_j(\beta) \vdash c_k : \gamma}{\vdash run_{i \cap j \cap k}(\gamma)} \; E+$$

$$\frac{\vdash run_i(\alpha) \qquad x_i : \alpha \vdash run_j(\beta)}{run_{i \cap j}(\alpha \supset \beta)} \; I \supset \; \frac{\vdash run_{i \cap j}(\alpha \supset \beta) \qquad x_i : \alpha}{x_i : \alpha \vdash run_j(\beta)} \; E \supset$$

Connectives II

$$\frac{\vdash exec(\alpha) \qquad a_i : \alpha \vdash exec(\beta)}{\vdash run_{i \cup j}(\alpha \to \beta)} \; I \to$$

$$\frac{\vdash run_{i \cup j}(\alpha \to \beta) \qquad a_i : \alpha}{exec(\alpha) \vdash exec(\beta)} \; E \to$$

$$\frac{\vdash run_{i \cap j}(\alpha \supset \beta) \qquad a_i : \alpha}{\vdash synchro_j(\beta(exec(\alpha)))} \; \text{Synchro}$$

A Premise: Local means Mobile

- Locally valid terms can be made mobile;
- Errors occur for terms that can be fetched, accessed, broadcasted;
- Our calculus will admit *mobility*.

Step 2: Add Modal Functions for Mobility

- *GLOB*: is a $\Box$-Intro rule; it says that an *exec* value is everywhere valid;
- *BROAD*: is a $\Diamond$-Intro rule; it says that a *run* term can be transmitted to a given location;
- *RET*: is a $\Box$-Elim rule; it says that a *GLOB* value can be returned to a specific address;
- *SEND*: is a $\Diamond$-Elim rule; it moves a *run* term from a place to another.

Modified Syntax

Definition (Syntax)

Mobility is given as:

$$\texttt{Remote Operations} := \mathit{GLOB}(\Box_{i\cup j}\Gamma, \alpha) \mid \mathit{BROAD}(\Diamond_{i\cap j}\Gamma, \alpha)$$

$$\texttt{Portable Code} := \mathit{RET}(\Gamma_{i\cup j}, \alpha) \mid \mathit{SEND}(\Gamma_{i\cap j}, \alpha)$$

Rules for Modality (Intro & Elimination)

$$\frac{\Gamma_i, x_j : \alpha \vdash run_j(\alpha) \qquad \Box_i\Gamma, x_j(a_j) : \alpha \vdash exec(\alpha)}{GLOB(\Box_{i\cup j}\Gamma, \alpha)}\ \text{RPC1}$$

$$\frac{\Gamma_i, x_j : \alpha \vdash run_j(\alpha) \qquad \Diamond_i\Gamma \vdash run_j(\alpha)}{BROAD(\Diamond_{i\cap j}\Gamma, \alpha)}\ \text{RPC2}$$

$$\frac{\Box_i\Gamma, a_j : \alpha \vdash exec(\alpha) \qquad GLOB(\Box_{i\cup j}\Gamma, \alpha)}{RET(\Gamma_{i\cup j}, \alpha)}\ \text{PORT1}$$

$$\frac{\Box_i\Gamma, x_j : \alpha \vdash run_{i\cap j}(\alpha) \qquad BROAD(\Diamond_{i\cap j}\Gamma, \alpha)}{SEND(\Gamma_{i\cap j}, \alpha)}\ \text{PORT2}$$

A Premise: Which Errors

- We consider two main kinds of errors:
  - Mistakes; semantic or syntactic errors, generated by non-valid terms at locations;
  - Failures: procedural errors, generated by failing mobility instructions.

Step 3: Add Error Functions

Two kinds of errors:

1. *fail*$@_i(\tau)$: the state of the system where a term of type τ induces a failure when accessed at index i;

2. *mistake*(τ): the state of the system where reference to type τ induces an error.

Step 3: Add Error Functions

Two kinds of errors:

1. *fail*$@_i(\tau)$: the state of the system where a term of type τ induces a failure when accessed at index i;
2. *mistake*(τ): the state of the system where reference to type τ induces an error.

Two concurrency functions:

1. *access*$@_i(t : \tau \setminus \bot)$: a command to access a term at a given address;
2. $\phi(\tau)$*WITH*$(\Pi \cup \mathrm{I} \cup \Gamma)$: a command to execute a resource $\phi := \{run, exec\}$ with a local term;

Modified Syntax

Definition (Syntax)

Errors are defined/use:

$\texttt{H}(\texttt{Error Functions}) := \textit{fail}@_i(t) \mid \textit{mistake}(\tau)$

$\texttt{C}(\texttt{Concurrency Functions}) := \textit{access}@_i(t) \mid \phi(\tau)\textit{WITH}(\Pi \cup I \cup \Gamma)$

Rules for Concurrency

$$\frac{\Delta_i, a_i : \alpha \vdash exec(\alpha)}{\Delta_i \vdash access@_i(a : \alpha)} \text{@I} \qquad \frac{\Delta_i; \Gamma_i \vdash access@_i(a : \alpha)}{\Delta_i; \Gamma_i, x_i : \alpha \vdash run_i(\alpha)} \text{@E}$$

$$\frac{\Delta_i; \Gamma_i \vdash \phi(\alpha)}{\Delta_i \vdash \phi(\alpha) WITH(t, \Gamma)} \text{WITH-I}$$

$$\frac{\Delta_i \vdash \phi(\alpha) WITH(t, \Gamma) \qquad \Delta_i; \Gamma \vdash t : \alpha}{\Delta_i; \Gamma, t : \alpha \vdash \phi(\alpha)} \text{WITH-E}$$

Mistakes: semantic errors

Mistake1: a dependency from a locally invalid value

$$\frac{\Delta_i; \Gamma_i \vdash run_i(\tau \rightarrow \bot) \qquad \Delta_i; x_i : \tau \vdash run_i(\upsilon)}{\Gamma_i; \Delta_i \vdash mistake(exec(\upsilon) WITH(\tau))} \text{ Mistake1}$$

Mistakes: semantic errors

Mistake1: a dependency from a locally invalid value

$$\frac{\Delta_i;\Gamma_i \vdash run_i(\tau \rightarrow \bot) \qquad \Delta_i; x_i : \tau \vdash run_i(\upsilon)}{\Gamma_i;\Delta_i \vdash mistake(exec(\upsilon)WITH(\tau))} \text{ Mistake1}$$

HandleMistake1: re-define dependency from local resource:

$$\frac{\Gamma_i \vdash mistake(exec(\upsilon)WITH(\tau)) \qquad \Gamma_i, t_j : \tau' \vdash exec(\tau')}{\Gamma_i \vdash synchro_j(\upsilon(exec([\tau/\tau'])))} \text{ HandleMistake1}$$

Mistakes: syntax error

Mistake2: dependency from a locally invalid term

$$\frac{\Delta_i; \Gamma_i \vdash run_i(t \supset \bot) \qquad \Gamma_i, x_i : \tau \vdash run_i(\upsilon)}{\Gamma_i; \Delta_i \vdash mistake(run_i(\upsilon) WITH(t_i))} \text{ Mistake2}$$

Mistakes: syntax error

Mistake2: dependency from a locally invalid term

$$\frac{\Delta_i; \Gamma_i \vdash run_i(t \supset \bot) \qquad \Gamma_i, x_i : \tau \vdash run_i(\upsilon)}{\Gamma_i; \Delta_i \vdash mistake(run_i(\upsilon) WITH(t_i))} \text{ Mistake2}$$

HandleMistake2: re-define dependency from required resource

$$\frac{\Gamma_i \vdash mistake(run_i(\upsilon) WITH(t_i)) \qquad \Gamma_i, [t_i/x_j] : \tau \vdash run_i(\upsilon)}{\Gamma_i \vdash run_{j \cap i}(\tau \supset \upsilon)} \text{ HandleMistake2}$$

Failures: access wrong resources

Failure1: access wrong resources possibly at right location

$$\frac{\Box_i\Gamma, t_j : \tau \vdash exec(\upsilon) \qquad GLOB(\Box_{i\cup j}\Gamma, \tau)}{\Box_i\Gamma, access@_j(t' : \tau') \vdash fail@_{i\cup j}(\upsilon) \quad (t' \neq t; \tau' \neq \tau)} \text{FailPort1}$$

Failures: access wrong resources

Failure1: access wrong resources possibly at right location

$$\frac{\Box_i\Gamma, t_j : \tau \vdash exec(\upsilon) \qquad GLOB(\Box_{i\cup j}\Gamma, \tau)}{\Box_i\Gamma, access@_j(t' : \tau') \vdash fail@_{i\cup j}(\upsilon) \quad (t' \neq t; \tau' \neq \tau)} \text{ FailPort1}$$

HandleFailure1: re-execute access to local resource

$$\frac{\Box_i\Gamma, access@_j(t' : \tau') \vdash fail@_{i\cup j}(\upsilon) \qquad RET(\Gamma_{i\cup j}, \tau) \qquad \tau \neq \tau'}{\Box_i\Gamma, [t'_j : \tau' / t_j : \tau] \vdash exec(\upsilon)} \text{ HFP1}$$

Failures: access wrong locations

Failure2: access right resources at wrong locations

$$\frac{\Box_i\Gamma; x_j : \tau \vdash run_{i\cap j}(\upsilon) \qquad BROAD(\Diamond_{i\cap j}\Gamma, \tau)}{\Diamond_i\Gamma, access@_{k>j}(t : \tau) \vdash fail@_{i\cap j}(\upsilon)} \text{ FailPort2}$$

Failures: access wrong locations

Failure2: access right resources at wrong locations

$$\frac{\Box_i\Gamma; x_j : \tau \vdash run_{i\cap j}(\upsilon) \qquad BROAD(\Diamond_{i\cap j}\Gamma, \tau)}{\Diamond_i\Gamma, access@_{k>j}(t : \tau) \vdash fail@_{i\cap j}(\upsilon)} \text{ FailPort2}$$

HandleFailure2: re-execute access to local resource

$$\frac{\Diamond_i\Gamma, access@_{k>j}(t : \tau) \vdash fail@_{i\cap j}(\upsilon) \qquad SEND(\Gamma_{i\cap j}, \tau)}{\Diamond_i\Gamma, [t_k/x_j] : \tau \vdash run_{i\cap j}(\upsilon)} \text{ HFP2}$$

Some interesting theorems

- Local Soundness and Completeness: Error Rules and Handling Rules behave as Introduction and Elimination Rules; by normalization, one obtains an equivalent rule without detour;

Some interesting theorems

- Local Soundness and Completeness: Error Rules and Handling Rules behave as Introduction and Elimination Rules; by normalization, one obtains an equivalent rule without detour;
- Structural: Weakening, Contraction and Exchange for Mistake and Failure expressions are provable;

Some interesting theorems

- Local Soundness and Completeness: Error Rules and Handling Rules behave as Introduction and Elimination Rules; by normalization, one obtains an equivalent rule without detour;
- Structural: Weakening, Contraction and Exchange for Mistake and Failure expressions are provable;
- Reductions: Equivalent or Inducible Errors can be substituted preserving Error states; β-reductions and η-expansions hold;

Some interesting theorems

- Local Soundness and Completeness: Error Rules and Handling Rules behave as Introduction and Elimination Rules; by normalization, one obtains an equivalent rule without detour;
- Structural: Weakening, Contraction and Exchange for Mistake and Failure expressions are provable;
- Reductions: Equivalent or Inducible Errors can be substituted preserving Error states; β-reductions and η-expansions hold;
- Termination: either an error state or its handling is at some point a non-further reducible expression (based on non-iteration of errors).

Philosophical Remarks

1. Validity is here intended as a local notion;
2. Errors originate from such a restricted understanding of validity and from the additional use of the notion of accessibility;
3. Handling is always a procedural request to reformulate the correct version of a given process.

Related Work

1. Data Quality as defined by Errors elimination;
2. Definition of Distrust and Mistrust Propagation by Error Production in information channels; with application to Expertise;
3. Future Work: Risk, Doubt, Consensus reaching strategies by error resolution.

Thanks

- Enough to think about? Then, Thanks!
- Want to see more? Semantics is waiting for you after this slide...

A Semantics of Transitions

- Satisfaction: define the operations that allow to transit to the state of the system where that expression holds; for every satisfiable expression of the system there is such a transition;
- Validity: for every transition, there is a final state that is reached; at such state, every valid expression holds;
- Correctness: one needs to show that a satisfiable transition does not invalidate an expression it moves from.

Transitions

Definition (Operational Model)

An operational model of the procedural semantics for the machine is a model where each S is evaluated by transition to some S'. An indexed transition system, called a `Network`

$$\texttt{Network} := (\mathcal{S}, \mapsto, \mathcal{I})$$

is a triple with $\mathcal{S} \subseteq \texttt{States}, \mathcal{I} \subseteq \texttt{Indices}$ and $\mapsto$ a ternary relation over indexed states $(\mathcal{S} \times \mathcal{I} \times \mathcal{S})$. If $S, S' \in \mathcal{S}$ and $i, j \in \mathcal{I}$, then $\mapsto (S, i, j, S')$ is written as $S_i \mapsto S'_j$. This means that there is a transition $\mapsto$ from state S valid at index i to state S' valid at index j defined according to the machine typing rules.

The Domain of Correct Expressions

Definition (Standard Domain)

$$SD := \{S \mid \exists S' : (S, S') \in \texttt{Network} \; \& \; (S \mapsto S') : \alpha\}$$

Moving in SD

Definition (Transitions of the Standard Domain)

The rewriting of a state S into a valid state $S' \in SD$ is established by the following transitions:

	$S \mapsto S'$
run	$(\Gamma_i, x_i : \alpha) \mapsto (\Diamond_i \Gamma, run_i(\alpha))$
exec	$(\Gamma_i, a_i : \alpha) \mapsto (\Box_i \Gamma, exec(\alpha))$
corun	$(\Gamma_i, run_i(\alpha) \vdash b_j : \beta) \mapsto (\Box_i \Gamma, run_{i \cap j}(\alpha(\beta))$
coexec	$(\Gamma_i, exec(\alpha) \vdash b_j : \beta) \mapsto (\Box_i \Gamma, run_{i \cup j}(\alpha(\beta))$
synchro	$(\Box_i \Gamma, run_{i \cup j}(\alpha(\beta)) \mapsto (\Box_i \Gamma, synchro_j(\beta(exec(\alpha))))$
product	$(\Gamma_i, exec(\alpha), exec(\beta)) \mapsto (\Box_i \Gamma, run_{i \cap j}(\alpha \times \beta))$
extraction1	$(\Box_i \Gamma, run_{i \cap j}(\alpha \times \beta)) \mapsto (\Box_i \Gamma, exec(\alpha))$
extraction2	$(\Box_i \Gamma, run_{i \cap j}(\alpha \times \beta)) \mapsto (\Box_i \Gamma, exec(\beta))$
tagunion	$(\Gamma_i, exec(\alpha)) \mapsto (\Box_i \Gamma, run_{i \cup j}(\alpha + \beta))$
patternmatch1	$(\Box_i \Gamma, run_{i \cup j}(\alpha + \beta) \vdash c_k : \gamma) \mapsto (\Box_i \Gamma, run_{i \cap k}(\alpha(\gamma)))$
patternmatch2	$(\Box_i \Gamma, run_{i \cup j}(\alpha + \beta) \vdash c_k : \gamma) \mapsto (\Box_i \Gamma, run_{j \cap k}(\beta(\gamma)))$
$\Box$1	$(\Box_i \Gamma, exec(\alpha)) \mapsto (GLOB(\Box_{i \cup j} \Gamma, \alpha))$
$\Box$2	$(\Box_{i \cup j} \Gamma, \alpha) \mapsto (RET(\Gamma_{i \cup j}, \alpha))$
$\Diamond$1	$(\Diamond_i \Gamma, run_j(\alpha)) \mapsto (BROAD(\Diamond_{i \cap j} \Gamma, \alpha))$
$\Diamond$2	$(\Diamond_{i \cap j} \Gamma, \alpha) \mapsto (SEND(\Gamma_{i \cap j}, \alpha))$

The Semantics of Error Expressions

Definition (Exit Network)

An indexed exit transition system called an `ExitNetwork`

$$\texttt{ExitNetwork} := (\mathcal{S}, \mapsto_e, \mathcal{I})$$

is a triple with $\mathcal{S} \subseteq \texttt{States}, \mathcal{I} \subseteq \texttt{Indices}$ and $\mapsto_e$ a quaternary relation over indexed states $(\mathcal{S} \times \mathcal{I} \times \mathcal{S} \times E)$, with E the set of all declared exit points. If $S, S' \in \mathcal{S}$ and $i, j \in \mathcal{I}$, then $\mapsto (S, i, j, S', e)$ is written as $S_i \mapsto S_j \mapsto e$. This means that there is a transition $\mapsto$ from state S valid at index i to state S' invalid at index j defined according to the machine typing rules which leads to an exit point e.

The Semantics of Error Expressions

Definition (Failure Domain)

$$FD := \{S \mid \exists S' : (S, S') \in \texttt{ExitNetwork} \;\&\; S \notin SD \cup \texttt{Network}\}$$

or in other words $(S \mapsto S' \mapsto e) : \neg\alpha$.

Moving in FD

Definition (Transitions of the Failure Domain)

The rewriting of a state $S \in SD$ into an invalid state $S' \in FD$ is established by the following transitions:

	$S \mapsto S'$
mistake1	$(\Diamond_i \Gamma, run_i(\tau(\bot)) \vdash exec(\upsilon)) \mapsto (\Box_i \Gamma, mistake(exec(\tau(\upsilon))))$
mistake2	$(\Diamond_i \Gamma, run_i(t(\bot)) \vdash run_i(\upsilon)) \mapsto (\Diamond_i \Gamma, mistake(run_i(\tau(\upsilon))))$
failport1	$(GLOB(\Box_{i \cup j} \Gamma, \tau(\upsilon))) \mapsto (RET(\Box_{i \cup j} \Gamma, fail(t'(\upsilon))))$
failport2	$(BROAD(\Diamond_{i \cap j} \Gamma, (\tau(\upsilon)))) \mapsto (SEND(\Box_{i \cap k > j} \Gamma, fail(t(\upsilon))))$

Moving to Resolve

Definition (Resolve Transitions)

The rewriting of an invalid state $S \in FD$ into a valid state $S' \in SD$ is established by the following transitions:

	$S \mapsto^{res} S'$
resolveM1	$(\Box_i\Gamma, mistake(exec(\tau(\upsilon)))) \mapsto^{res} (\Box_i\Gamma, run_{i\cup j}(\tau'(\upsilon))$
resolveM2	$(\Diamond_i\Gamma, mistake(run_i(\tau(\upsilon))) \mapsto^{res} (\Box_i\Gamma, run_{i\cap j}(\tau'(\upsilon))$
resolveFP1	$(RET(\Box_{i\cup j}\Gamma, fail(t(\upsilon)))) \mapsto^{res} (\Box_i\Gamma, exec(\tau') \vdash u_j : \upsilon)$
resolveFP2	$(SEND(\Box_{i\cap k>j}\Gamma, fail(t(\upsilon)))) \mapsto^{res} (\Box_i\Gamma, run_j(\tau) \vdash u_j : \upsilon)$

Moving to Abort

Definition (Abort Transitions)

The final abort state is a valid state $S \in SD$ established by the following transitions:

	$S \mapsto^{abort} S'$
abortM1	$(GLOB(\Box_i\Gamma, mistake(exec(\tau(\upsilon))))) \mapsto^{ab} (\Gamma_{i\cup j}, abort(\tau(\upsilon))$
abortM2	$(BROAD(\Diamond_i\Gamma, mistake(run_i(\tau(\upsilon)))) \mapsto^{ab} (\Gamma_{i\cap j}, abort(\tau(\upsilon))$
abortFP1	$(\Box_{i\cup j}\Gamma, fail(t(\upsilon))) \mapsto^{ab} (\Gamma_{i\cup j}, abort(\tau(\upsilon))$
abortFP2	$(\Box_{i\cap k>j}\Gamma, fail(t(\upsilon))) \mapsto^{ab} (\Gamma_{i\cap k}, abort(\tau(\upsilon)))$

Some interesting theorems

Theorem (Progress)

For every state in SD, either there is a transition or its type is the output value. For every state in FD, either there is a resolution transition or an abort transition and the type of the latter is the output value.

Some interesting theorems

Theorem (Progress)

For every state in SD, either there is a transition or its type is the output value. For every state in FD, either there is a resolution transition or an abort transition and the type of the latter is the output value.

Theorem (Preservation)

For every state in SD, its transition is type-preserving. For every state in FD, every resolution transition is type-preserving w.r.t. another state and every abort transition is of output value.

Some interesting theorems

Theorem (Progress)

For every state in SD, either there is a transition or its type is the output value. For every state in FD, either there is a resolution transition or an abort transition and the type of the latter is the output value.

Theorem (Preservation)

For every state in SD, its transition is type-preserving. For every state in FD, every resolution transition is type-preserving w.r.t. another state and every abort transition is of output value.

Theorem (Type Safety)

The `ExitNetwork` *semantics is safe: for every state in SD, there is either a type-preserving transition or the state provides the output value; for every state in FD, there is either a type-preserving transition to a resolved state or the* $\bot$*-type provides the output value.*

References I

Allchin, D. (2001).
Error types.
Perspectives on Science, 9:38–59.

Bonnay, D. and Egre', P. (2011).
Knowing One's Limits - An analysis in Centered Dynamic Epistemic Logic.
Synthese, Springer.

Cristian, F. (1982).
Exception handling and software fault-tolerance.
IEEE Transactions on Computers, C-31:531–540.

Dekker, S. (2011).
Drift into Failure.
Ashgate.

Goodenough, J. (1975).
Exception handling: issues and a proposed notation.
Commun. ACM, 8:683–696.

References II

Mayo, D. (1996).
Error and the Growth of Experimental Knowledge.
Chicago University Press.

Mayo, D. (2010).
Learning from error, severe testing, and the growth of theoretical knowledge.
In Mayo, D. and Spanos, editors, *Error and Inference*. Cambridge University Press.

Primiero, G. (2013).
A taxonomy of errors for information systems.
Minds & Machines.

Reason, J. (1990).
Human Error.
Cambridge University Press.

References III

- Sundholm, B. (2012).
  Error.
  Topoi.
- Turner, R. (2011).
  Specification.
  Minds & Machines, 21(2).:135–152.
- Williamson, T. (1992).
  On intuitionistic modal epistemic logic.
  Journal of Philosophical Logic, 21:63–89.
- Williamson, T. (2002).
  Knowledge and its Limits.
  Oxford University Press.
- Woods, D.D, D. S. C. R. J. L. S. N., editor (2010).
  Behind Human Error.
  Ashgate.

References IV

Woods, H. (2004).
The Death of Argument: Fallacies in Agent-based Reasoning.
luwer Academic Publishers.