Formal Verification of Software

Dynamic Logic for Java

Bernhard Beckert

UNIVERSITÄT KOBLENZ-LANDAU

Summer Term 2005

KeY Supports Java Card as Target Language

What is Java Card?

- Subset of Java
- Sun's official standard for SMARTCARDS and embedded devices

KeY Supports Java Card as Target Language

What is Java Card?

- Subset of Java
- Sun's official standard for SMARTCARDS and embedded devices

Why Java Card?

KeY Supports Java Card as Target Language

What is Java Card?

- Subset of Java
- Sun's official standard for SMARTCARDS and embedded devices

Why Java Card?

Good example for real-world object-oriented language

KeY Supports Java Card as Target Language

What is Java Card?

- **Subset of Java**
- **Sun's official standard for SMARTCARDS and embedded devices**

Why Java Card?

Good example for real-world object-oriented language

Java Card has *no*

- **garbage collection**
- **dynamical class loading**
- **multi-threading**
- **floating-point arithmetic**

KeY Supports Java Card as Target Language

What is Java Card?

- **Subset of Java**
- **Sun's official standard for SmartCards and embedded devices**

Why Java Card?

Good example for real-world object-oriented language

Java Card has *no*

- **garbage collection**
- **dynamical class loading**
- **multi-threading**
- **floating-point arithmetic**

Application Area

- **security critical**
- **financial risk (e.g. exchanging smart cards is expensive)**

Academic vs. Real-world Languages

Problems to address

Pointers / objects attributes

Modelled as non-rigid constants and functions

Academic vs. Real-world Languages

Problems to address

Pointers / objects attributes

Modelled as non-rigid constants and functions

Side effects

Expressions in programs have side effects, for example

```
if ((y=3) + y < 0) .. else ..
```

Academic vs. Real-world Languages

Problems to address

Pointers / objects attributes

Modelled as non-rigid constants and functions

Side effects

Expressions in programs have side effects, for example

```
if ((y=3) + y < 0) .. else ..
```

Aliasing

Different names may refer to the same location, for example

$o.a, u.a$ **in a state** g **where** $g \models o \doteq u$

Other Issues (Later)

Further supported Java Card features

- **method invocation, dynamic binding**

Other Issues (Later)

Further supported Java Card features

- **method invocation, dynamic binding**
- **polymorphism**

Other Issues (Later)

Further supported Java Card features

- **method invocation, dynamic binding**
- **polymorphism**
- **abrupt termination**

Other Issues (Later)

Further supported Java Card features

- **method invocation, dynamic binding**
- **polymorphism**
- **abrupt termination**
- **checking for nullpointer exceptions**

Other Issues (Later)

Further supported Java Card features

- **method invocation, dynamic binding**
- **polymorphism**
- **abrupt termination**
- **checking for nullpointer exceptions**
- **object creation and initialisation**

Other Issues (Later)

Further supported Java Card features

- **method invocation, dynamic binding**
- **polymorphism**
- **abrupt termination**
- **checking for nullpointer exceptions**
- **object creation and initialisation**
- **arrays**

Other Issues (Later)

Further supported Java Card features

- **method invocation, dynamic binding**
- **polymorphism**
- **abrupt termination**
- **checking for nullpointer exceptions**
- **object creation and initialisation**
- **arrays**
- **finiteness of integer data types**

Other Issues (Later)

Further supported Java Card features

- **method invocation, dynamic binding**
- **polymorphism**
- **abrupt termination**
- **checking for nullpointer exceptions**
- **object creation and initialisation**
- **arrays**
- **finiteness of integer data types**
- **transactions**

Handling Object Attributes

Similar concepts

- Object attributes
- Arrays
- Pointers

Handling Object Attributes

Similar concepts

- Object attributes
- Arrays
- Pointers

Non-rigid functions

Attributes are considered to be non-rigid functions on objects

Handling Object Attributes

Similar concepts

- Object attributes
- Arrays
- Pointers

Non-rigid functions

Attributes are considered to be non-rigid functions on objects

Extended to program variables

Program variables are considered to be non-rigid constants

Side Effects: Symbolic Execution Paradigm

Expressions may have side effects, for example a simple assignment

```
(y=3) + y < 0
```

does not only evaluate to a `boolean` **value, but also assigns a value to** `y`**.**

Side Effects: Symbolic Execution Paradigm

Expressions may have side effects, for example a simple assignment

`(y=3) + y < 0`

does not only evaluate to a `boolean` **value, but also assigns a value to** `y`**.**

Problem: Terms in logic have to be side effect free

Side Effects: Symbolic Execution Paradigm

Expressions may have side effects, for example a simple assignment

`(y=3) + y < 0`

does not only evaluate to a `boolean` value, but also assigns a value to `y`.

Problem: Terms in logic have to be side effect free

Solution:

- **Calculus rules realise a stepwise symbolic execution of the programs (program transformation)**

Side Effects: Symbolic Execution Paradigm

Expressions may have side effects, for example a simple assignment

`(y=3) + y < 0`

does not only evaluate to a `boolean` value, but also assigns a value to `y`.

Problem: **Terms in logic have to be side effect free**

Solution:

- **Calculus rules realise a stepwise symbolic execution of the programs (program transformation)**
- **Restrict applicability of some rules. For example, `if-then-else` is only applicable, if the guard is free of side-effects**

Rule Application for `if-then-else`

$$\Gamma \vdash \langle \texttt{if}\ ((\texttt{y} = 3)\ +\ \texttt{y}\ <\ \texttt{0})\{\alpha\}\ \texttt{else}\{\beta\}\rangle\, \Phi, \Delta$$

Rule Application for `if-then-else`

$$\frac{\Gamma \vdash \langle \texttt{boolean guard} \;=\; (\texttt{y} = \texttt{3}) \;+\; \texttt{y} \;<\; \texttt{0};\;\; \texttt{if}\;(\texttt{guard})\{\alpha\}\;\texttt{else}\{\beta\}\;\rangle\,\Phi,\;\Delta}{\Gamma \vdash \langle \texttt{if}\;((\texttt{y} = \texttt{3}) \;+\; \texttt{y} \;<\; \texttt{0})\{\alpha\}\;\texttt{else}\{\beta\}\rangle\,\Phi,\;\Delta}$$

Rule Application for `if-then-else`

$$\frac{\dfrac{\Gamma \vdash \left\langle \begin{array}{l} \texttt{int val0} \;=\; (\texttt{y} = 3) \;+\; \texttt{y;} \\ \texttt{boolean guard} \;=\; \texttt{val0} \;<\; \texttt{0;} \\ \texttt{if (guard)}\{\alpha\}\ \texttt{else}\{\beta\} \end{array} \right\rangle \Phi,\ \Delta}{\Gamma \vdash \langle \texttt{boolean guard} \;=\; (\texttt{y} = 3) \;+\; \texttt{y} \;<\; \texttt{0;}\ \ \texttt{if (guard)}\{\alpha\}\ \texttt{else}\{\beta\}\ \rangle\, \Phi,\ \Delta}}{\Gamma \vdash \langle \texttt{if ((y} = 3) \;+\; \texttt{y} \;<\; \texttt{0)}\{\alpha\}\ \texttt{else}\{\beta\}\rangle\, \Phi, \Delta}$$

Rule Application for `if-then-else`

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \left\langle \begin{array}{l} \texttt{int val1} \;=\; \mathrm{y} = 3; \\ \texttt{int val0} \;=\; \texttt{val1} \;+\; \mathrm{y} \\ \ldots \end{array} \right\rangle \Phi,\ \Delta}{\Gamma \vdash \left\langle \begin{array}{l} \texttt{int val0} \;=\; (\mathrm{y} = 3) \;+\; \mathrm{y}; \\ \texttt{boolean}\ \mathrm{guard} \;=\; \texttt{val0} \;<\; 0; \\ \texttt{if}\ (\mathrm{guard})\{\alpha\}\ \texttt{else}\{\beta\} \end{array} \right\rangle \Phi,\ \Delta}}{\Gamma \vdash \langle \texttt{boolean}\ \mathrm{guard} \;=\; (\mathrm{y} = 3) \;+\; \mathrm{y} \;<\; 0;\ \ \texttt{if}\ (\mathrm{guard})\{\alpha\}\ \texttt{else}\{\beta\}\ \rangle\, \Phi,\ \Delta}}{\Gamma \vdash \langle \texttt{if}\ ((\mathrm{y} = 3) \;+\; \mathrm{y} \;<\; 0)\{\alpha\}\ \texttt{else}\{\beta\} \rangle\, \Phi, \Delta}$$

Rule Application for `if-then-else`

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \left\langle \begin{array}{l} \texttt{y} = \texttt{3;} \\ \texttt{int val1} = \texttt{y;} \\ \texttt{int val0} = \texttt{val1} + \texttt{y} \\ \ldots \end{array} \right\rangle \Phi,\ \Delta}{\Gamma \vdash \left\langle \begin{array}{l} \texttt{int val1} = \texttt{y} = \texttt{3;} \\ \texttt{int val0} = \texttt{val1} + \texttt{y} \\ \ldots \end{array} \right\rangle \Phi,\ \Delta}}{\Gamma \vdash \left\langle \begin{array}{l} \texttt{int val0} = (\texttt{y} = \texttt{3}) + \texttt{y;} \\ \texttt{boolean guard} = \texttt{val0} < \texttt{0;} \\ \texttt{if (guard)}\{\alpha\}\ \texttt{else}\{\beta\} \end{array} \right\rangle \Phi,\ \Delta}}{\Gamma \vdash \langle \texttt{boolean guard} = (\texttt{y} = \texttt{3}) + \texttt{y} < \texttt{0;}\ \texttt{if (guard)}\{\alpha\}\ \texttt{else}\{\beta\} \rangle \Phi,\ \Delta}}{\Gamma \vdash \langle \texttt{if ((y} = \texttt{3}) + \texttt{y} < \texttt{0)}\{\alpha\}\ \texttt{else}\{\beta\}\rangle \Phi, \Delta}$$

Assignment in the Classical Version

Classical rule for assignment

$$\frac{\Gamma^{x \leftarrow y}, x \doteq t^{x \leftarrow y} \quad \vdash \quad \Phi,\ \Delta^{x \leftarrow y}}{\Gamma \quad \vdash \quad \langle x = t \rangle \Phi,\ \Delta} \ (y \textit{ new variable})$$

Assignment in the Classical Version

Classical rule for assignment

$$\frac{\Gamma^{x \leftarrow y}, x \doteq t^{x \leftarrow y} \quad \vdash \quad \Phi,\ \Delta^{x \leftarrow y}}{\Gamma \quad \vdash \quad \langle x = t \rangle \Phi,\ \Delta} \; (y \textit{ new variable})$$

Problems:

- ***cannot*** **be handled as substitution**

Assignment in the Classical Version

Classical rule for assignment

$$\frac{\Gamma^{x \leftarrow y}, x \doteq t^{x \leftarrow y} \;\vdash\; \Phi,\ \Delta^{x \leftarrow y}}{\Gamma \;\vdash\; \langle x = t \rangle \Phi,\ \Delta} \ (y \textit{ new variable})$$

Problems:

- ***cannot*** **be handled as substitution**
- **aliasing:**

$$\frac{?}{o.a \doteq 3 \;\vdash\; \langle u.a = 5;\ \rangle \phi}$$

Requires to split the proof for the cases $o = u$ **and** $o \neq u$**.**

The Active Statement in a Program

Example

$$\underbrace{\texttt{l:\{try\{}}_{\pi}\ \texttt{i=0;}\ \underbrace{\texttt{j=0; \} finally\{ k=0; \}\}}}_{\omega}$$

first active command	`i=0;`
non-active prefix	π
rest	ω

Updates: Delayed Substitutions

Syntax: Updates are syntactical elements

$$\{loc := val\}\Phi \text{ or } \{loc := val\}t$$

where

loc **either a**

- **program variable** x
- **an attribute** $o.attr$ **or**
- **an array access** $a[i]$

val **a logical term (no side effects)**

Updates: Delayed Substitutions

Syntax: Updates are syntactical elements

$$\{loc := val\}\Phi \text{ or } \{loc := val\}t$$

where

loc **either a**

- **program variable** x
- **an attribute** $o.attr$ **or**
- **an array access** $a[i]$

val **a logical term (no side effects)**

Semantic:

$$g \models \{loc := val\}\Phi \qquad \textbf{iff} \qquad g' \models \Phi \textbf{ where } g' = g_{loc}^{val}$$

Assignment Rule in KeY

$$\frac{\Gamma \vdash \{\mathtt{loc} := \mathtt{val}\}\langle \pi\ \omega \rangle \Phi,\ \Delta}{\Gamma \vdash \langle \pi\ \mathtt{loc} = \mathtt{val};\ \omega \rangle \Phi,\ \Delta}$$ **, where** $loc,\ val$ **side effect free**

Assignment Rule in KeY

$$\frac{\Gamma \vdash \{\texttt{loc} := \texttt{val}\}\langle \pi\ \omega \rangle \Phi,\ \Delta}{\Gamma \vdash \langle \pi\ \texttt{loc} = \texttt{val};\ \omega \rangle \Phi,\ \Delta}$$, **where** $loc,\ val$ **side effect free**

Advantages:

- **no renaming as in the classical version**

Assignment Rule in KeY

$$\frac{\Gamma \vdash \{\texttt{loc} := \texttt{val}\}\langle \pi\ \omega \rangle \Phi,\ \Delta}{\Gamma \vdash \langle \pi\ \texttt{loc} = \texttt{val};\ \ \omega \rangle \Phi,\ \Delta}$$, **where** $loc,\ val$ **side effect free**

Advantages:

- **no renaming as in the classical version**
- **delayed proof branching**

$$\Gamma \vdash \langle \mathtt{x} = 3;\ \ \mathtt{x} = 4;\ \rangle \Phi$$ **or**

$$\Gamma \vdash \langle \mathtt{o.a} = 3;\ \ \mathtt{o.a} = 4;\ \rangle \Phi$$

Conditional Terms

Use conditional terms to delay splitting further

$$(s[t_1 \; ?= t_2] \mapsto e)^{I,\beta} \quad = \quad \begin{cases} e^{I,\beta} & t_1^{I,\beta} = t_2^{I,\beta} \\ (s[t_1])^{I,\beta} & \textbf{otherwise} \end{cases}$$

Application of updates $\mathcal{U}$

Application on

program variable

$$\{x := t\}\, y \quad \leadsto \quad y$$

$$\{x := t\}\, x \quad \leadsto \quad t$$

$$\{o.a := t\}\, y \quad \leadsto \quad y$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\, y \rightsquigarrow y$$

$$\{x := t\}\, x \rightsquigarrow t$$

$$\{o.a := t\}\, y \rightsquigarrow y$$

Application on attribute

$$\{o.a := t\}\, o.a \rightsquigarrow t$$

$$\{o.a := t\}\, u.a \rightsquigarrow (\{o.a := t\}u? = o).a \mapsto t$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\, y \;\rightsquigarrow\; y$$

$$\{x := t\}\, x \;\rightsquigarrow\; t$$

$$\{o.a := t\}\, y \;\rightsquigarrow\; y$$

Application on attribute

$$\{o.a := t\}\, o.a \;\rightsquigarrow\; t$$

$$\{o.a := t\}\, u.a \;\rightsquigarrow\; (\{o.a := t\}u? = o).a \mapsto t$$

Application stops before modal operators, e.g.

$$\{o.a := t\}\, \langle\alpha\rangle\Phi \;\rightsquigarrow\; \{o.a := t\}\, \langle\alpha\rangle\Phi$$

Application is shoved over operators to the subformulas (terms)

$$\{o.a := t\}\, \Phi \wedge \Psi \;\rightsquigarrow\; \{o.a := t\}\Phi \wedge \{o.a := t\}\Psi$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\, y \quad \rightsquigarrow \quad y$$

$$\{x := t\}\, x \quad \rightsquigarrow \quad t$$

$$\{o.a := t\}\, y \quad \rightsquigarrow \quad y$$

Application on attribute

$$\{o.a := t\}\, o.a \quad \rightsquigarrow \quad t$$

$$\{o.a := t\}\, u.a \quad \rightsquigarrow \quad (\{o.a := t\}u? = o).a \mapsto t$$

Example

$$\{o.a := o\}o.a.a.b$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\, y \quad \rightsquigarrow \quad y$$

$$\{x := t\}\, x \quad \rightsquigarrow \quad t$$

$$\{o.a := t\}\, y \quad \rightsquigarrow \quad y$$

Application on attribute

$$\{o.a := t\}\, o.a \quad \rightsquigarrow \quad t$$

$$\{o.a := t\}\, u.a \quad \rightsquigarrow \quad (\{o.a := t\}u? = o).a \mapsto t$$

Example

$$\{o.a := o\}o.a.a.b \quad \rightsquigarrow \quad \{o.\color{blue}{a} := o\}o.a.a.\color{red}{b}$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\, y \quad \rightsquigarrow \quad y$$

$$\{x := t\}\, x \quad \rightsquigarrow \quad t$$

$$\{o.a := t\}\, y \quad \rightsquigarrow \quad y$$

Application on attribute

$$\{o.a := t\}\, o.a \quad \rightsquigarrow \quad t$$

$$\{o.a := t\}\, u.a \quad \rightsquigarrow \quad (\{o.a := t\}u? = o).a \mapsto \; t$$

Example

$$\{o.a := o\}o.a.a.b \quad \rightsquigarrow \quad (\{o.{\color{blue}a} := o\}o.a.{\color{blue}a}).b$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\, y \quad \leadsto \quad y$$

$$\{x := t\}\, x \quad \leadsto \quad t$$

$$\{o.a := t\}\, y \quad \leadsto \quad y$$

Application on attribute

$$\{o.a := t\}\, o.a \quad \leadsto \quad t$$

$$\{o.a := t\}\, u.a \quad \leadsto \quad (\{o.a := t\}u? = o).a \mapsto \; t$$

Example

$$\{o.a := o\}o.a.a.b \quad \leadsto \quad (\,(\{o.a := o\}o.a? = o).a \mapsto \; o).b$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\, y \quad \leadsto \quad y$$

$$\{x := t\}\, x \quad \leadsto \quad t$$

$$\{o.a := t\}\, y \quad \leadsto \quad y$$

Application on attribute

$$\{o.a := t\}\, o.a \quad \leadsto \quad t$$

$$\{o.a := t\}\, u.a \quad \leadsto \quad (\{o.a := t\}u ?= o).a \mapsto t$$

Example

$$\{o.a := o\}o.a.a.b \quad \leadsto \quad (\,(o ?= o).a \mapsto o).b$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\ y \quad \rightsquigarrow \quad y$$

$$\{x := t\}\ x \quad \rightsquigarrow \quad t$$

$$\{o.a := t\}\ y \quad \rightsquigarrow \quad y$$

Application on attribute

$$\{o.a := t\}\ o.a \quad \rightsquigarrow \quad t$$

$$\{o.a := t\}\ u.a \quad \rightsquigarrow \quad (\{o.a := t\}u ? = o).a \mapsto \ t$$

Example

$$\{o.a := o\}o.a.a.b \quad \rightsquigarrow \quad (\ (o ? = o).a \mapsto \ o).b$$

Application of updates $\mathcal{U}$

Application on program variable

$$\{x := t\}\, y \;\rightsquigarrow\; y$$

$$\{x := t\}\, x \;\rightsquigarrow\; t$$

$$\{o.a := t\}\, y \;\rightsquigarrow\; y$$

Application on attribute

$$\{o.a := t\}\, o.a \;\rightsquigarrow\; t$$

$$\{o.a := t\}\, u.a \;\rightsquigarrow\; (\{o.a := t\}u? = o).a \mapsto t$$

Example

$$\{o.a := o\}o.a.a.b \;\rightsquigarrow\; o.b$$

Parallel Updates

Computing update followed by update

$$\{l_1 := r_1\}\{l_2 := r_2\} \;=\; \{\{l_1 := r_1\},\ \{\{l_1 := r_1\} \downarrow l_2 := \{l_1 := r_1\}r_2\}\}$$

where $\mathcal{U} \downarrow l = \begin{cases} x & \textbf{if } l = x \textbf{ is a program variable} \\ (\mathcal{U}\, u).a & \textbf{if } l = u.a \end{cases}$

Results in parallel update: $\{l_1 := v_1,\ \ldots,\ l_n := v_n\}$

Parallel Updates

Computing update followed by update

$$\{l_1 := r_1\}\{l_2 := r_2\} \;=\; \{\{l_1 := r_1\},\ \{\{l_1 := r_1\} \downarrow l_2 := \{l_1 := r_1\}r_2\}\}$$

where $\mathcal{U} \downarrow l = \begin{cases} x & \textbf{if } l = x \textbf{ is a program variable} \\ (\mathcal{U}\, u).a & \textbf{if } l = u.a \end{cases}$

Results in parallel update: $\{l_1 := v_1,\ \ldots,\ l_n := v_n\}$

Semantics

- **All l_i and v_i computed in old state**
- **All updates done simultaneously**
- **If conflct** $l_i = l_j$**,** $v_i \neq v_j$ **later update wins**

Quantifying over Program Variables

Cannot quantify over program variables (non-ridig constants)

Non allowed: $\forall i{:}\mathtt{int}\,(\langle \alpha(i) \rangle F)$

Non allowed: $\forall n\,(\langle \alpha(n) \rangle F)$

Quantifying over Program Variables

Cannot quantify over program variables (non-ridig constants)

Non allowed: $\forall\, i{:}\texttt{int}\, (\langle \alpha(i) \rangle F)$

Non allowed: $\forall\, n\, (\langle \alpha(n) \rangle F)$

Solution

$$\forall\, n\, \{i := n\} \langle \alpha(i) \rangle F)$$

Abrupt Changes of the Control Flow

Abrupt Termination: **Redirection of the control fbw by**

`return`, `break`, `continue` **or** `Exceptions`

Abrupt Changes of the Control Flow

Abrupt Termination: **Redirection of the control flow by**

`return`, `break`, `continue` **or** `Exceptions`

$\langle \texttt{try} \{$

$\quad \texttt{a} = \texttt{a}/\texttt{b};$

$\quad \texttt{a} = \texttt{a} + \texttt{1};$

$\} \ \texttt{catch(Exception e)} \ \{\ldots\}$

$\texttt{finally} \ \{\ldots\} \rangle \ \Phi$

Decomposition Rule

not applicable

Abrupt Changes of the Control Flow

Abrupt Termination: **Redirection of the control fbw by**

$$\texttt{return},\ \texttt{break},\ \texttt{continue}\ \textbf{or}\ \texttt{Exceptions}$$

$$\langle \texttt{try}\{$$
$$\texttt{a} = \texttt{a}/\texttt{b};$$
$$\texttt{a} = \texttt{a} + \texttt{1};$$
$$\}\ \texttt{catch}(\texttt{Exception e})\ \{\ldots\}$$
$$\texttt{finally}\ \{\ldots\}\rangle\ \Phi$$

Decomposition Rule

not applicable

Solution: **The rules work on the first active statement**

$$\frac{\Gamma \vdash \langle \pi\ stmnt';\ \omega\rangle\Phi,\ \Delta}{\Gamma \vdash \langle \pi\ stmnt;\ \omega\rangle\Phi,\ \Delta}$$

Catch Thrown Exception

Rule

$$\frac{}{\Gamma \ \vdash \ \langle \texttt{try}\{\ \texttt{throw}\ \texttt{exc;}\ \ p\} \\ \texttt{catch}\ (\texttt{Exception}\ \texttt{e})\ \{q\} \\ \texttt{finally}\{r\}\rangle\ \Phi,\ \Delta}$$

Catch Thrown Exception

Rule

$$\frac{\Gamma \ \vdash \ \langle \texttt{if}\ (\texttt{exc}\ \texttt{instanceof}\ \texttt{Exception})\ \{ \ \texttt{try}\{\texttt{e} = \texttt{exc};\ \ q\}\ \texttt{finally}\{r\} \ \}\ \texttt{else}\ \{\ r\ \texttt{throw}\ \texttt{exc};\ \}\rangle\Phi,\ \Delta}{\Gamma \ \vdash \ \langle \texttt{try}\{\ \texttt{throw}\ \texttt{exc};\ \ p\} \ \texttt{catch}\ (\texttt{Exception}\ \texttt{e})\ \{q\} \ \texttt{finally}\{r\}\rangle\ \Phi,\ \Delta}$$