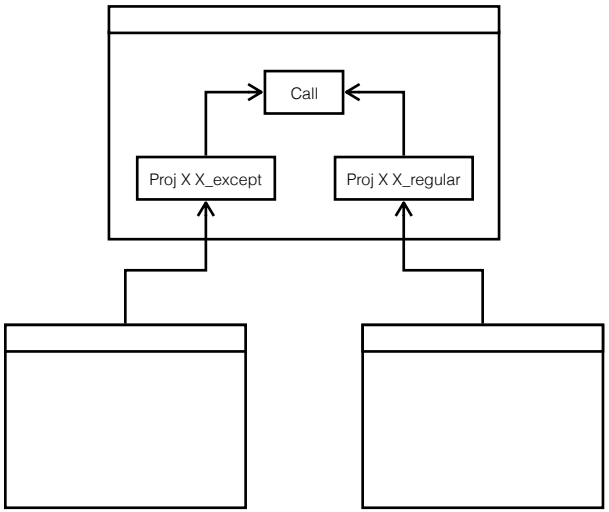
Exception Support in a Graph-Based Intermediate Representation

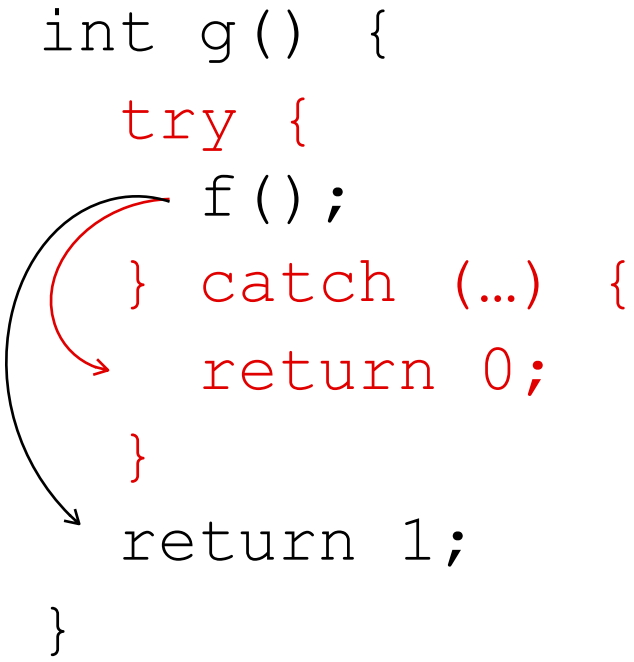
Bachelorarbeit von Jonas Haag



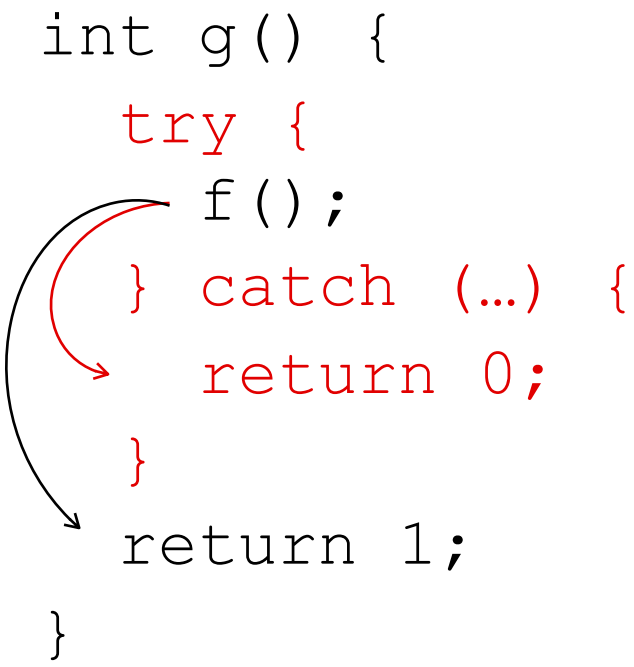
Einleitung

```
int g() {

    f();



  return 1;
}
```

Einleitung



Einleitung



- **Zwischenrepräsentation**
  Modellierung neuer Steuerfluss
  Implementierung im Middle-End
- **Frontend**
  `catch`-Blöcke erzeugen
- **Backend**
  Neuer Steuerfluss bei Codeerzeugung
- **Laufzeit**
  Richtigen Handler finden, Tabellen

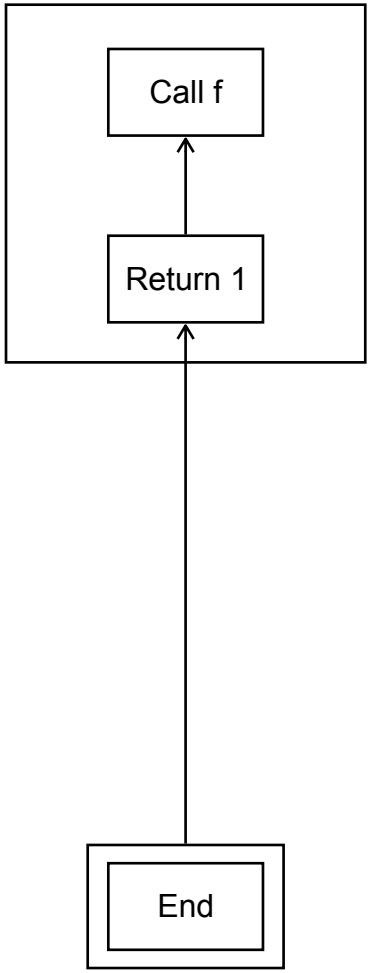
Einleitung

```
int g() {
  try {
    f();
  } catch (…) {
    return 0;
  }
  return 1;
}
```

- **Zwischenrepräsentation**
  Modellierung neuer Steuerfluss (A)
  Implementierung im Middle-End (B)
- **Frontend**
  `catch`-Blöcke erzeugen
- **Backend**
  Neuer Steuerfluss bei Codeerzeugung
- **Laufzeit**
  Richtigen Handler finden, Tabellen

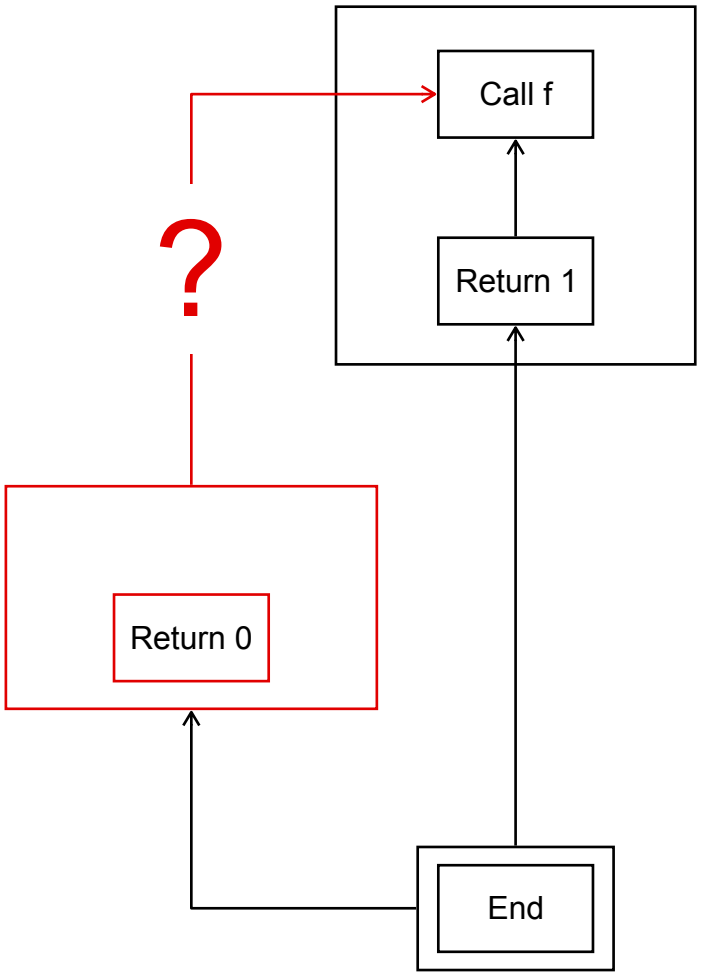
Teil A: Modellierung neuer Steuerfluss

```
int g() {
  try {
    f();
  } catch (…) {
    return 0;
  }
  return 1;
}
```



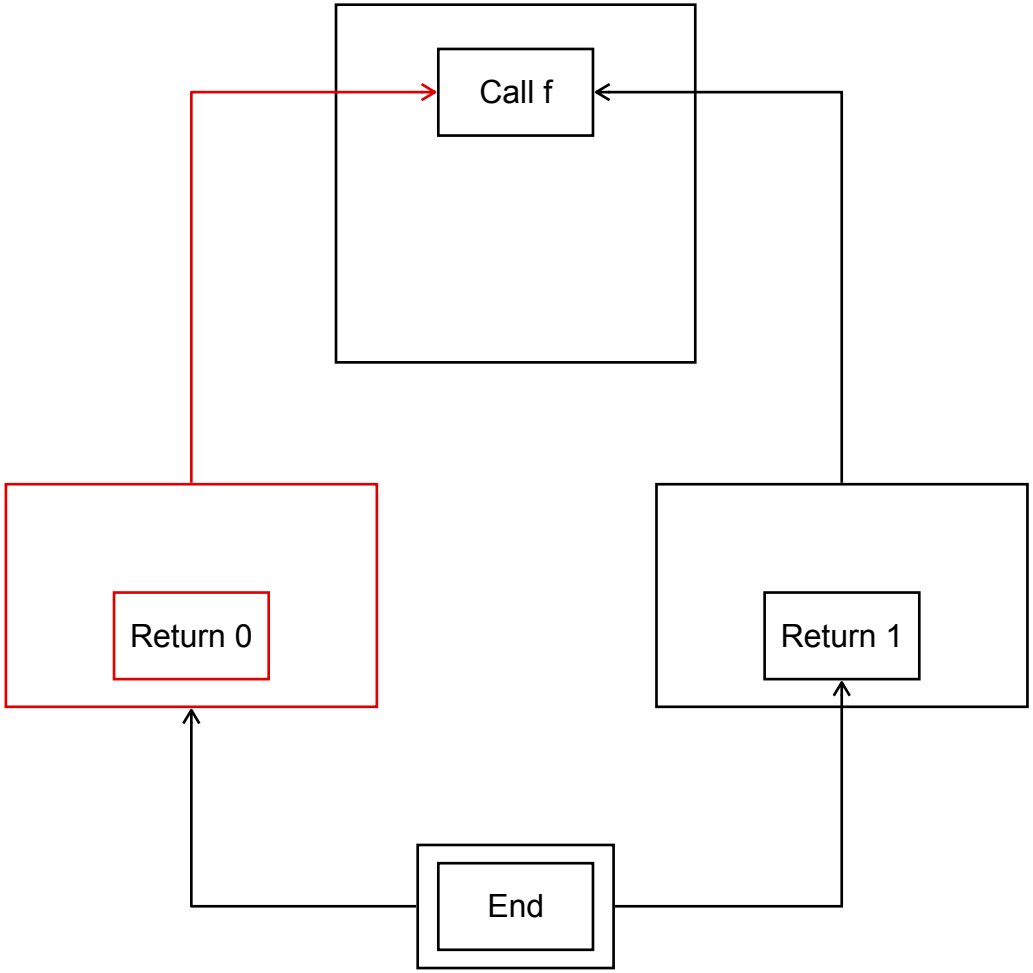
Teil A: Modellierung neuer Steuerfluss

```
int g() {
  try {
    f();
  } catch (…) {
    return 0;
  }
  return 1;
}
```



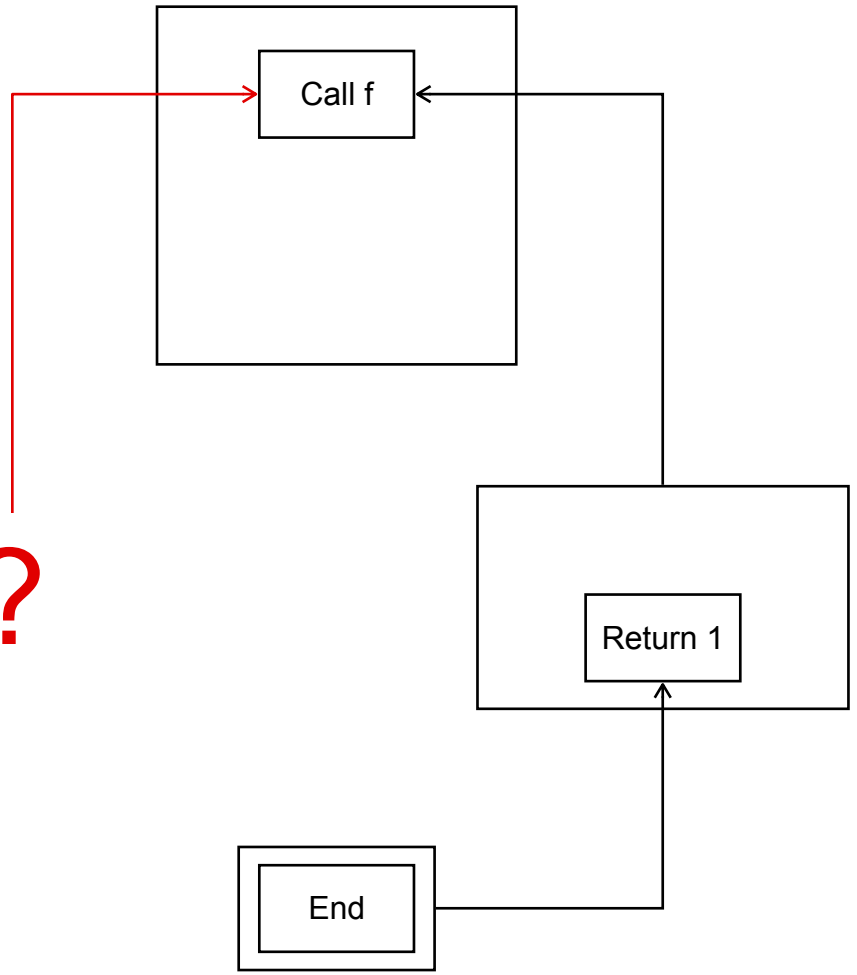
Teil A: Modellierung neuer Steuerfluss

```
int g() {
  try {
    f();
  } catch (…) {
    return 0;
  }
  return 1;
}
```



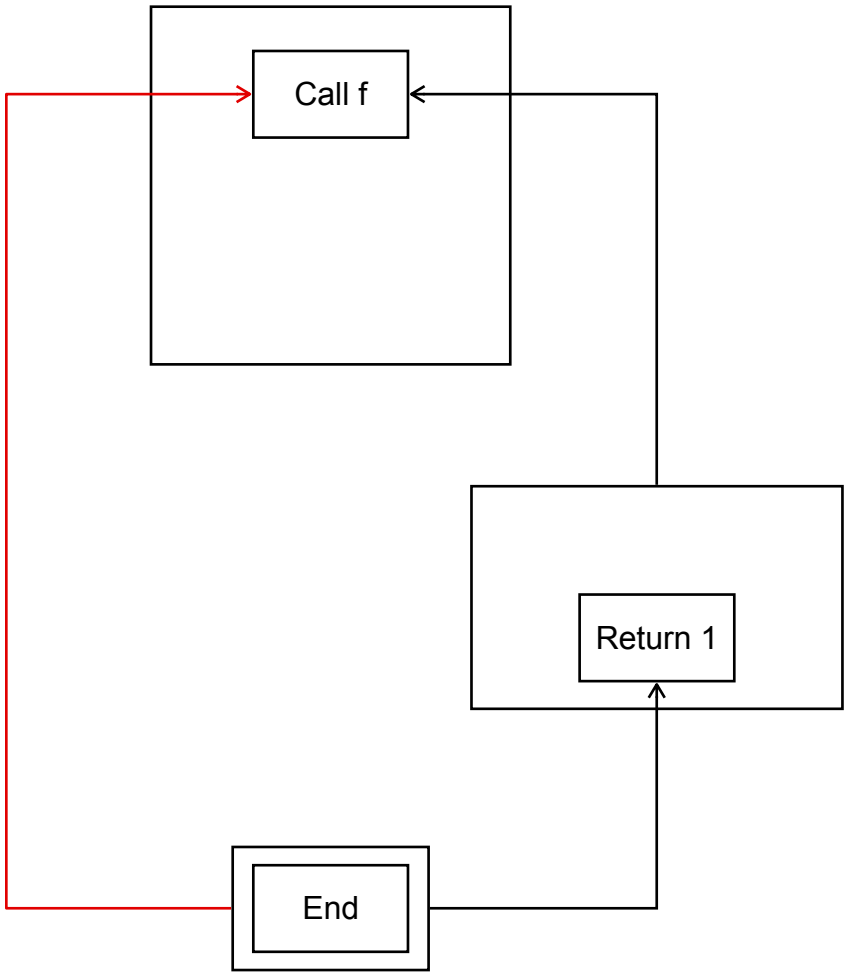
Teil A: „Entweichende” Ausnahmen

```
int g() {
  f();
  return 1;
}

void h() {
  try {
    g();
  } catch (…) {
    …
  }
}
```



Teil A: „Entweichende” Ausnahmen

```
int g() {
  f();
  return 1;
}

void h() {
  try {
    g();
  } catch (…) {
    …
  }
}
```

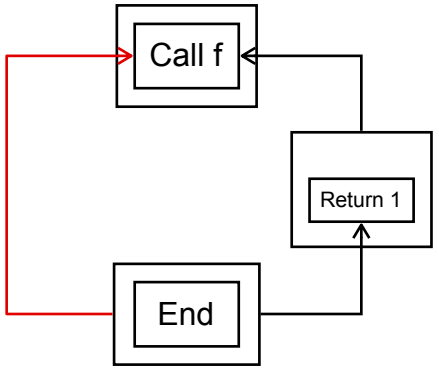


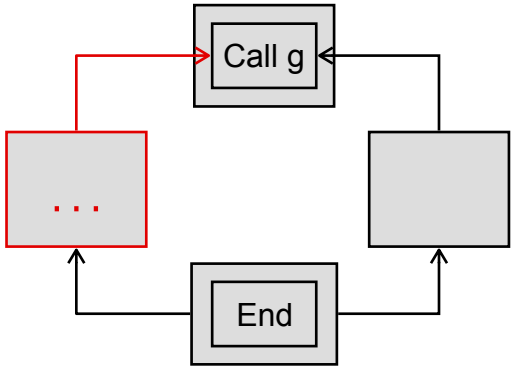
Teil A: Zusammenfassung

- Explizite Darstellung des Ausnahme-Steuerflusses
- Werfende Operation beendet Grundblock
- Beeinflusst Middle-End, Backend
  - z.B. Inline-Optimierung

Teil B: Inline-Optimierung

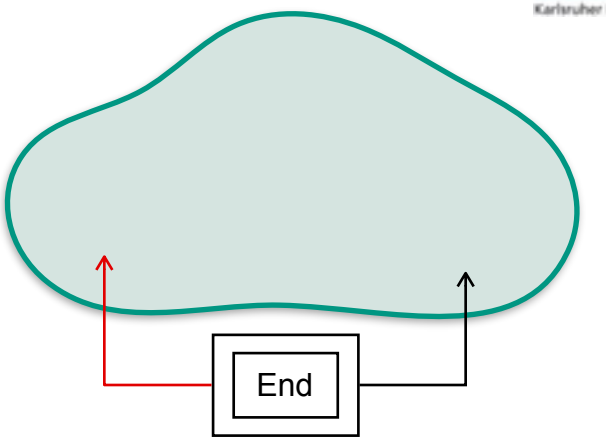
```
int g() {
  f();
  return 1;
}

void h() {
  try {
    g();
  } catch (…) {
    …
  }
}
```

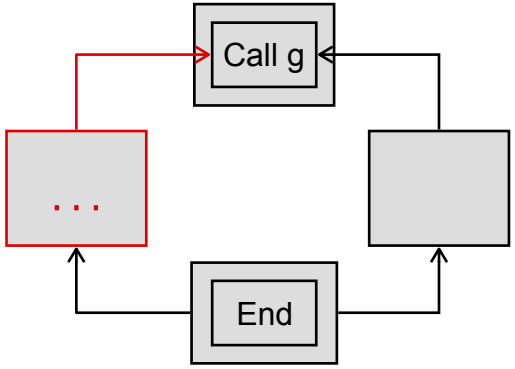




Teil B: Inline-Optimierung

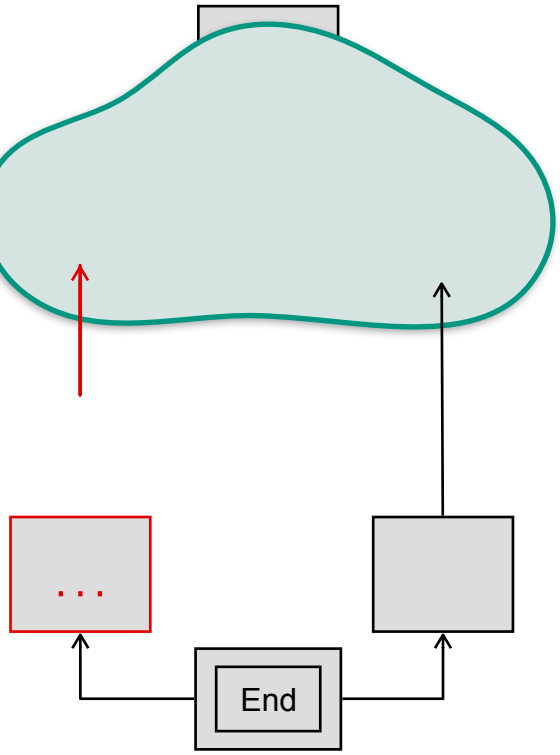
```
int g() {
   f();
   return 1;
}

void h() {
   try {
      g();
   } catch (…) {
      …
   }
}
```





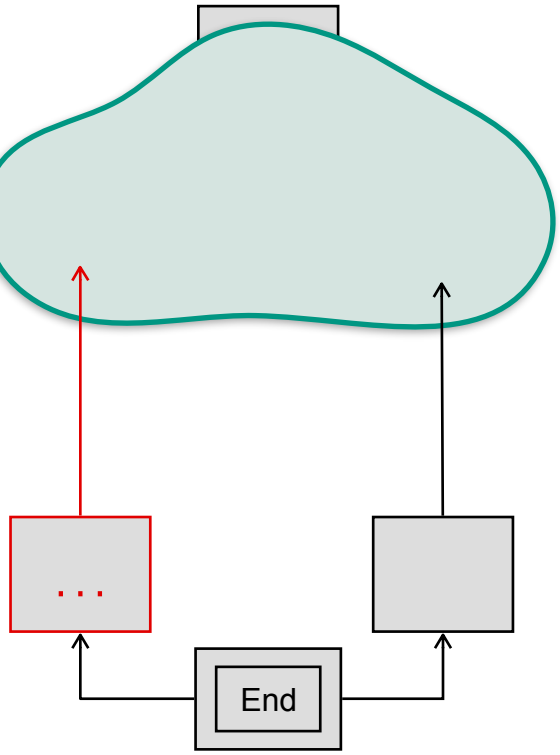
Teil B: Inline-Optimierung

```
int g() {
   f();
   return 1;
}

void h() {
   try {
      g();
   } catch (…) {
      …
   }
}
```



Teil B: Inline-Optimierung

```
int g() {
  f();
  return 1;
}

void h() {
  try {
    g();
  } catch (…) {
    …
  }
}
```



Teil B: Übersicht Inline-Optimierung

- Regulärer Steuerfluss wird in Block gebündelt
- Ausnahme-Steuerfluss führt in äußeres `catch` (oder direkt zum Ende)

Evaluierung

- Java, C
- Für C: Nimm für unbekannte Funktionen an, dass sie Ausnahmen werfen
- Werfende Operation beendet Grundblock, Erwartungen:
  - mehr Grundblöcke im Graphen
  - Schlechtere Optimierung
- Statische Analyse: Anzahl Instr., Grundblöcke, Instr. pro Grundblock
- Benchmarking mit SPEC2000-Suite

Evaluierung

- C:
  - Grundblöcke 1,5% kleiner
  - SPEC2000-Suite 0,6% langsamer

- Java:
  - Grundblöcke 240% kleiner
  - Einfluss auf Performance noch offen

Übersicht Bachelorarbeit

- **Zwischenrepräsentation**
  Werfende Operation beendet Grundblock
  Anpassung der Implementierung des Middle-Ends
- **Frontend**
  Java (BYTECODE2FIRM), C (CPARSER)
- **Backend**
  Firm-Knoten nach werfender Operation
  Block Scheduling
  Callee-Save-Register
  X87-Simulator
- **Laufzeit**
  Richtigen Handler durch Stack-Unwinding finden
  Tabellen: Abbildung werfende Operation → `catch`-Block

BACKUP

Umfang der Änderungen (Git)

	Files changed	Insertions	Deletions
LIBFIRM	47	1220	630
CPARSER	7	279	176
BYTECODE2 FIRM	4	85	38
LIBOO	4	57	54

Backend

- Ausnahme-Kontrollfluss beachten
- Block Scheduling (zusätzliche `jmps`)
- Knoten nach werfender Operation
- X87-Simulator
- Call-Frame-Information, enthält Information zu Registerzustand beim Stack-Unwinding
  - Stackframe
  - Callee-Save-Register

Laufzeit (1)

- Ausnahme-Kontrollfluss i.A. nicht lokal (sondern über Methodengrenzen)
- Muss zur Auswahl des `catch`-Blocks Call-Stack ablaufen
  - libunwind
  - Call-Frame-Information

Laufzeit (2)

- Auswahl des `catch`-Blocks benötigt Typchecks zur Laufzeit (Subtyping)

```
catch (MyException e) {
   …
}
```