Corten: Refinement Types for Imperative Languages with Ownership

Abschlusspräsentation Masterarbeit

Carsten Csiky | 26th Oktober 2022

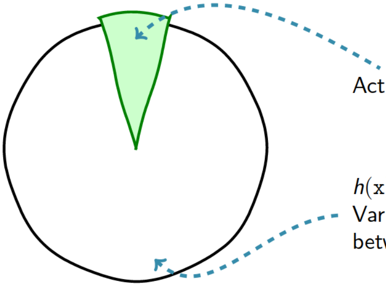
Bitte Logo über `\grouplogo{}` festlegen.

Inhaltsverzeichnis

1. Motivation
2. Type System
3. Soundness Justification
4. Related Work
5. Conclusion / Future Work

Motivation

```
public IntList square(IntList list) {
  return list.map(x -> x*x);
}
```



Motivation

Karlsruher Institut für Technologie

```rust
fn max(a: i32, b: i32) {
  if a > b { a } else { b }
}
```

Motivation

Karlsruher Institut für Technologie

```
fn max(a: i32, b: i32) {
  if a > b { a } else { b }
}
```

- Return Value (v) : $v \geq a \wedge v \geq b$

Motivation

Karlsruher Institut für Technologie

```rust
fn max(a: i32, b: i32) {
  if a > b { a } else { b }
}
```

- Return Value (v) : $v \geq a \wedge v \geq b$
- Rondon et al. [RKJ08]: Refinement Types for Functional Programming Languages

Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
  if a > b { a } else { b }
}
```

Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
  if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \land v \geq b\}$

$$\frac{}{\Gamma \vdash \texttt{if}\ a > b\ \{a\}\ \texttt{else}\ \{b\} : \tau}$$

Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
  if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \land v \geq b\}$

$$\dfrac{\dfrac{}{\Gamma, a > b \vdash a : \tau} \qquad \dfrac{}{\Gamma, \neg(a > b) \vdash b : \tau}}{\Gamma \vdash \texttt{if } a > b\ \{a\}\ \texttt{else}\ \{b\} : \tau}$$

Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
  if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

$$\dfrac{\dfrac{\dfrac{}{\Gamma, a > b \vdash \{v : \texttt{i32} \mid v \doteq a\} \preceq \tau}}{\Gamma, a > b \vdash a : \tau} \qquad \dfrac{}{\Gamma, \neg(a > b) \vdash b : \tau}}{\Gamma \vdash \texttt{if}\ a > b\ \{a\}\ \texttt{else}\ \{b\} : \tau}$$

Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
  if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \land v \geq b\}$

$$\dfrac{\dfrac{\dfrac{\star}{\Gamma, a > b \vdash a : \{v : \texttt{i32} \mid v \doteq a\}} \qquad \dfrac{}{\Gamma, a > b \vdash \{v : \texttt{i32} \mid v \doteq a\} \preceq \tau}}{\Gamma, a > b \vdash a : \tau} \qquad \dfrac{}{\Gamma, \neg(a > b) \vdash b : \tau}}{\Gamma \vdash \texttt{if } a > b \ \{a\} \texttt{ else } \{b\} : \tau}$$

Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
  if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

$$\dfrac{\dfrac{\dfrac{\star}{\Gamma, a > b \vdash a : \{v : \texttt{i32} \mid v \doteq a\}} \qquad \dfrac{\text{SMT-VALID}\begin{pmatrix}\text{true} \wedge \text{true} \wedge a > b \\ \wedge\, v \doteq a \\ \implies (v \geq a \wedge v \geq b)\end{pmatrix}}{\Gamma, a > b \vdash \{v : \texttt{i32} \mid v \doteq a\} \preceq \tau}}{\Gamma, a > b \vdash a : \tau} \qquad \dfrac{}{\Gamma, \neg(a > b) \vdash b : \tau}}{\Gamma \vdash \texttt{if } a > b\ \{a\}\ \texttt{else }\ \{b\} : \tau}$$

Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
  if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \land v \geq b\}$

$$
\dfrac{
\dfrac{\dfrac{\star}{\Gamma, a > b \vdash a : \{v : \texttt{i32} \mid v \doteq a\}}
\qquad
\dfrac{\text{SMT-VALID}\begin{pmatrix}\text{true} \land \text{true} \land a > b \\ \land\, v \doteq a \\ \implies (v \geq a \land v \geq b)\end{pmatrix}}{\Gamma, a > b \vdash \{v : \texttt{i32} \mid v \doteq a\} \preceq \tau}}
{\Gamma, a > b \vdash a : \tau}
\qquad
\dfrac{\vdots}{\Gamma, \neg(a > b) \vdash b : \tau}
}{\Gamma \vdash \texttt{if}\ a > b\ \{a\}\ \texttt{else}\ \{b\} : \tau}
$$

Motivation

Karlsruher Institut für Technologie

```rust
fn  clamp(a: &mut i32, b: i32) {
  if *a > b { *a = b }
}
```

Motivation

```
fn  clamp(a: &mut i32, b: i32) {
  if *a > b { *a = b }
}
fn  client(...) {
  ...
  clamp(&mut x, 5);
  clamp(&mut y, 6);
  print!(x);
  ...
}
```

Motivation

```
fn  clamp(a: &mut i32, b: i32) {
  if *a > b { *a = b }
}
fn  client(...) {
  ...
  clamp(&mut x, 5);
  clamp(&mut y, 6);
  print!(x);
  ...
}
```

What does this it `print(x)` output?

- In most imperative programming languages:
  - Could be: old `x` or `5`

Motivation

Karlsruher Institut für Technologie

```
fn  clamp(a: &mut i32, b: i32) {
  if *a > b { *a = b }
}
fn  client(...) {
  ...
  clamp(&mut x, 5);
  clamp(&mut y, 6);
  print!(x);
  ...
}
```

What does this it `print(x)` output?

- In most imperative programming languages:
  - Could be: old `x` or `5`
  - But also `6` (if x aliases with y)!

Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Motivation

```
fn  clamp(a: &mut i32, b: i32) {
  if *a > b { *a = b }
}
fn  client(...) {
  ...
  clamp(&mut x, 5);
  clamp(&mut y, 6);
  print!(x);
  ...
}
```

What does this it `print(x)` output?

- In most imperative programming languages:
  - Could be: old `x` or `5`
  - But also `6` (if x aliases with y)!
- In Rust:
  - Just old `x` or `5`
  - And nothing else!

Motivation

```
fn  clamp(a: &mut i32, b: i32) {
  // borrows a
  // owns b
  if *a > b { *a = b }
  // "returns" the borrow of a
}
fn  client(...) { // owns x, y
  ...
  clamp(&mut x, 5); // lend x mutably
  clamp(&mut y, 6); // lend y mutably
  print!(x);
  ...
}
```

Ownership in Rust: Mutability XOR Aliasing

Each lexical scope tracks permissions for visible memory objects. Possible Permission Levels:

- Owner (e.g. `b`)
  - can: read, write
  - transfer ownership (if no outstanding borrows)
- Mutable Reference (e.g. `&mut x`)
  - can: read, write
  - guarantee: no aliasing
- Immutable Reference (e.g. `&v`)
  - can: read, alias
  - guarantee: no mutation

Motivation

Consequences:

- unique data owner
- no global, mutable state
- no cycles in memory structure

Ownership in Rust: Mutability XOR Aliasing

Each lexical scope tracks permissions for visible memory objects. Possible Permission Levels:

- Owner (e.g. `b`)
  - can: read, write
  - transfer ownership (if no outstanding borrows)
- Mutable Reference (e.g. `&mut x`)
  - can: read, write
  - guarantee: no aliasing
- Immutable Reference (e.g. `&v`)
  - can: read, alias
  - guarantee: no mutation

Motivation

Consequences:

- unique data owner
- no global, mutable state
- no cycles in memory structure

Used for:

- safe non-gc memory management
- safe concurrency
- safe low-level hardware access
- ...

Ownership in Rust: Mutability XOR Aliasing

Each lexical scope tracks permissions for visible memory objects. Possible Permission Levels:

- Owner (e.g. `b`)
  - can: read, write
  - transfer ownership (if no outstanding borrows)
- Mutable Reference (e.g. `&mut x`)
  - can: read, write
  - guarantee: no aliasing
- Immutable Reference (e.g. `&v`)
  - can: read, alias
  - guarantee: no mutation

Motivation

Karlsruher Institut für Technologie

Consequences:

- unique data owner
- no global, mutable state
- no cycles in memory structure

Used for:

- safe non-gc memory management
- safe concurrency
- safe low-level hardware access
- ...
- ⇒ show: program verification as well

Ownership in Rust: Mutability XOR Aliasing

Each lexical scope tracks permissions for visible memory objects. Possible Permission Levels:

- Owner (e.g. `b`)
  - can: read, write
  - transfer ownership (if no outstanding borrows)
- Mutable Reference (e.g. `&mut x`)
  - can: read, write
  - guarantee: no aliasing
- Immutable Reference (e.g. `&v`)
  - can: read, alias
  - guarantee: no mutation

Refinement Types for Rust – Syntax

```rust
fn max(a: i32, b: i32) -> i32 {
  if a > b { a } else { b }
}
```

Addition of two macros

- `ty!`$\{l : b \mid \varphi\}$ in place of a type
- `relax_ctx!{ ... }` in place of a statement

Refinement Types for Rust – Syntax

```
fn max(
  a: ty!{ av: i32 | true },
  b: ty!{ bv : i32 | true }
) -> ty!{ v : i32 | v >= av && v >= bv } {
  if a > b { a } else { b }
}
```

Addition of two macros

- `ty!`$\{l : b \mid \varphi\}$ in place of a type
- `relax_ctx!{ ... }` in place of a statement

Refinement Types for Rust – Syntax

```
fn max(
  a: ty!{ av: i32 },
  b: ty!{ bv : i32 }
) -> ty!{ v : i32 | v >= av && v >= bv } {
  if a > b { a } else { b }
}
```

Addition of two macros

- `ty!`$\{l : b \mid \varphi\}$ in place of a type
- `relax_ctx!{ ... }` in place of a statement

Type Updates

```
fn decr() -> ty!{ w : i32 | w >= 0 } {
 let mut i = ... as ty!{ v: i32 | v > 0};
 i = i - 1;
 i
}
```

- Types need to change through execution
  - $\Rightarrow$ type updates
  - Separation of program-variables and logic-variables
  - Γ association of program- to logic-variables and predicate
  - $\Gamma \vdash s \Rightarrow \Gamma'$ (Statement Type Checking)
  - $\Gamma \vdash e : \tau$ (Expression Typing)
  - On assignment: *replace* association, *append* predicate

Type Updates

```
fn decr() -> ty!{ w : i32 | w >= 0 } {
 // Γ1 = ({} , true)
 let mut i = ... as ty!{ v: i32 | v > 0};
 // Γ2 = ({i ↦ v} , v > 0)
 i = i - 1;
 // Γ3 = ({i ↦ v2} , v > 0 ∧ v2 ≐ v − 1)
 i }
```

- Types need to change through execution
  - $\Rightarrow$ type updates
  - Separation of program-variables and logic-variables
  - Γ association of program- to logic-variables and predicate
  - $\Gamma \vdash s \Rightarrow \Gamma'$ (Statement Type Checking)
  - $\Gamma \vdash e : \tau$ (Expression Typing)
  - On assignment: *replace* association, *append* predicate

Type Updates

```
fn decr() -> ty!{ w : i32 | w >= 0 } {
 // Γ1 = ({} , true)
 let mut i = ... as ty!{ v: i32 | v > 0};
 // Γ2 = ({i ↦ v} , v > 0)
 i = i - 1;
 // Γ3 = ({i ↦ v2} , v > 0 ∧ v2 ≐ v − 1)
 i }
```

let $\Gamma_2 = \Gamma[i \mapsto v_1], v > 0$ and $\tau = \{v : \texttt{i32} \mid v > 0\}$

$$
\textsc{Seq}\ \dfrac{
\textsc{Decl}\ \dfrac{
\textsc{Intro-Sub}\ \dfrac{\Gamma_1 \vdash \ldots : \tau' \qquad \Gamma_1 \vdash \tau' \preceq \tau}{\Gamma_1 \vdash \texttt{...}\ \texttt{as}\ \tau : \tau}
}{\Gamma_1 \vdash \texttt{let i = ... as}\ \tau \Rightarrow \Gamma_2}
\qquad
\textsc{Ass}\ \dfrac{
\textsc{BinOp}\ \dfrac{\Gamma_1 \vdash v_2 \text{ fresh}}{\Gamma_1 \vdash \texttt{i - 1} : \{v_2 : \text{i32} \mid v_2 \doteq v - 1\}}
}{\Gamma_2 \vdash \texttt{i = i - 1} \Rightarrow \Gamma[i \mapsto v_2], v > 0, v_2 \doteq v - 1}
}{\Gamma_1 \vdash \texttt{let i = ... as}\ \tau\texttt{; i = i - 1} \Rightarrow \Gamma[i \mapsto v_2], v > 0, v_2 \doteq v - 1}
$$

Expression Typing $\Gamma \vdash e : \tau$

$$\textsc{Lit}\ \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash v : \{\alpha : b \mid \alpha \simeq [\![v]\!]\Gamma\}} \qquad \textsc{BinOp}\ \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash x_1 \odot x_2 : \{\alpha : b \mid \alpha \simeq [\![x_1 \odot x_2]\!]\Gamma\}}$$

$$\textsc{Var}\ \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash x : \{\alpha : b \mid \alpha \simeq [\![x]\!]\Gamma\}} \qquad \textsc{Intro-Sub}\ \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash \tau \preceq \tau'}{\Gamma \vdash e \ \texttt{as}\ \tau' : \tau'}$$

Statement Type Checking $\Gamma \vdash s \Rightarrow \Gamma'$

$$\textsc{If}\ \frac{\Gamma, \Gamma(x) \doteq \texttt{true} \vdash s_t \Rightarrow \Gamma' \qquad \Gamma, \Gamma(x) \doteq \texttt{false} \vdash s_e \Rightarrow \Gamma'}{\Gamma \vdash \text{if } x \text{ then } s_t \text{ else } s_e \Rightarrow \Gamma'}$$

$$\textsc{Seq}\ \frac{\Gamma \vdash s_1 \Rightarrow \Gamma' \qquad \Gamma' \vdash s_2 \Rightarrow \Gamma''}{\Gamma \vdash s_1 ; s_2 \Rightarrow \Gamma''}$$

$$\textsc{Decl}\ \frac{\Gamma \vdash e : \{\beta : b \mid \varphi\}}{\Gamma \vdash \texttt{let}\ x = e \Rightarrow \Gamma[x \mapsto \beta], \varphi} \qquad \textsc{Assign}\ \frac{\Gamma \vdash e : \{\beta : b \mid \varphi\}}{\Gamma \vdash x = e \Rightarrow \Gamma[x \mapsto \beta], \varphi}$$

References – Strong Updates

```
fn client() -> ty!{ v: i32 | v == 4 } {
    let a = 2;        // a : {v₁ : i32 | v₁ == 2}
    let b = &mut a;   // b : {v₂ : &i32 | v₂ == &a}
    *b = 0; // changes a's value and type
    let c = &mut b;   // c : {v₃ : &i32 | v₃ == &b}
    **c = 4; // changes a's value and type
    a
}
```

Expression Typing $\Gamma \vdash e : \tau$

$$\textsc{Ref}\ \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash \&x : \{\alpha : \&b \mid \alpha \simeq [\![\&x]\!]\Gamma\}}$$

$$\textsc{Var-Deref}\ \frac{\Gamma \vdash *x \in \{y\} \qquad \Gamma \vdash y : \tau}{\Gamma \vdash *x : \tau}$$

Statement Type Checking $\Gamma \vdash s \Rightarrow \Gamma'$

$$\textsc{Assign-Strong}\ \frac{\Gamma(z) = \beta \qquad \Gamma \vdash *x \in \{y\} \qquad \Gamma \vdash \gamma \text{ fresh}}{\Gamma \vdash *x = z \Rightarrow \Gamma[y \mapsto \gamma], \gamma \doteq \beta}$$

Expression Typing $\Gamma \vdash e : \tau$

$$\textsc{Ref}\ \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash \&x : \{\alpha : \&b \mid \alpha \simeq [\![\&x]\!]\Gamma\}}$$

$$\textsc{Var-Deref}\ \frac{\Gamma \vdash *x \in \{y\} \qquad \Gamma \vdash y : \tau}{\Gamma \vdash *x : \tau}$$

Statement Type Checking $\Gamma \vdash s \Rightarrow \Gamma'$

$$\textsc{Assign-Strong}\ \frac{\Gamma(z) = \beta \qquad \Gamma \vdash *x \in \{y\} \qquad \Gamma \vdash \gamma \text{ fresh}}{\Gamma \vdash *x = z \Rightarrow \Gamma[y \mapsto \gamma], \gamma \doteq \beta}$$

$$\textsc{Assign-Weak}\ \frac{\dfrac{\Gamma \vdash e : \tau}{\Gamma \vdash y_i : \{\beta_i : b_i \mid \varphi_i\}} \qquad \dfrac{\Gamma \vdash *x \in \{y_1, \ldots, y_n\}}{\Gamma \vdash \tau \preceq \{\beta_i : b_i \mid \varphi_i\}}}{\Gamma \vdash *x = e \Rightarrow \Gamma}$$

Modularity & Mutation

```
fn clamp(a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 }, b: ty!{ b1: i32 }) {
  if *a > b { *a = b }
}
fn client(...) {
  ...
  clamp(&mut x, 5);
  clamp(&mut y, 6);
  print!(x);
  ...
}
```

Modularity & Mutation

```
fn clamp(a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 }, b: ty!{ b1: i32 }) {
  // $\Gamma_1 = (\{a \mapsto \&arg_0, arg_0 \mapsto a_1, b \mapsto b_1\}, \text{true} \wedge \text{true})$
  if *a > b { *a = b }
  // $\Gamma_2 = (\{arg_0 \mapsto a_2, b \mapsto b_1\}, a2 \leq b_1 \wedge \text{true})$
}
fn client(...) {
  ...
  // $\Gamma_1 = (\{x \mapsto v_1, y \mapsto v_2\}, \ldots)$
  clamp(&mut x, 5);
  // $\Gamma_2 = (\{x \mapsto v_3, y \mapsto v_2\}, \ldots \wedge v_3 \leq 5)$
  clamp(&mut y, 6);
  // $\Gamma_3 = (\{x \mapsto v_3, y \mapsto v_4\}, \ldots \wedge v_3 \leq 5 \wedge v_4 \leq 6)$
  print!(x);
  ...
}
```

Soundness

Progress

If $\Gamma \vdash s_1, \sigma : \Gamma \Rightarrow \Gamma_2$ and $s_1 \neq \texttt{unit}$, then there is a s_2 and σ_2 with $\langle s_1 \mid \sigma_1 \rangle \rightsquigarrow \langle s_2 \mid \sigma_2 \rangle$.

Corten strictly refines the base language, therefore progress depends on base type system.

Preservation

If $\Gamma \vdash s \Rightarrow \Gamma_2, \sigma : \Gamma$ and $\langle s \mid \sigma \rangle \rightsquigarrow \langle s_1 \mid \sigma_1 \rangle$, then there is a Γ_1 with $\Gamma_1 \vdash s_1 \Rightarrow \Gamma_2$ and $\sigma_2 : \Gamma_2$

Stronger property than base language preservation: Show that refined types are preserved

State Conformance

State Conformance $\sigma : \Gamma$

A state σ is conformant with respect to a typing context $\Gamma = (\mu, \Phi)$ (written as $\sigma : \Gamma$), iff:

$$\Phi[\mu(x) \triangleright [\![\sigma(x)]\!] \mid x \in \mathrm{dom}(\mu)] \text{ is satisfiable}$$

I.e. a conformant type context does not contradict the execution state.

Examples:

- If $\sigma : (\emptyset, \Phi)$ then Φ is satisfiable
- If $\sigma : (\mu, \Phi_1 \wedge \Phi_2)$ then $\sigma : (\mu, \Phi_1)$ and $\sigma : (\mu, \Phi_1)$.
- If $\sigma : (\mu, \Phi)$ and $\mathrm{FV}(\Phi) \subseteq \mathrm{dom}(\mu)$, then $\models \Phi[\mu(x) \triangleright [\![\sigma(x)]\!] \mid x \in \mathrm{dom}(\mu)]$

Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Intermediate Steps

Karlsruher Institut für Technologie

Conformance of Symbolic Execution

If $\sigma : \Gamma, \Gamma \vdash \alpha$ fresh then $\sigma[x \mapsto [\![e]\!]\sigma] : \Gamma[x \mapsto \alpha], (\alpha \simeq [\![e]\!]\Gamma)$

where $(\alpha \simeq [\![e]\!]\Gamma)$ is the symbolic execution of e equated with α in context Γ

Reference Predicates are Conservative

If $\sigma : \Gamma$ and $\Gamma \vdash *x \in \{y_1, \ldots, y_n\}$ then $[\![\sigma(x)]\!] = \&y_i$ for some $i \in 1, \ldots, n$

Rare case where conservative typing requires

Sub-Context Relation is Conservative

If $\Gamma \preceq \Gamma'$ and $\sigma : \Gamma$ then $\sigma : \Gamma'$

Literatur

Karlsruher Institut für Technologie

Backup-Teil

Folien, die nach `\beginbackup` eingefügt werden, zählen nicht in die Gesamtzahl der Folien.

[1] Patrick M. Rondon, Ming Kawaguci und Ranjit Jhala. „Liquid types“. In: *Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation - PLDI '08*. the 2008 ACM SIGPLAN conference. Tucson, AZ, USA: ACM Press, 2008, S. 159. ISBN: 978-1-59593-860-2. DOI: `10.1145/1375581.1375602`. URL: `http://portal.acm.org/citation.cfm?doid=1375581.1375602` (besucht am 30. 06. 2022).

Blöcke

in den KIT-Farben

Greenblock

Standard (`block`)

Blueblock

= `exampleblock`

Redblock

= `alertblock`

Brownblock

Purpleblock

Cyanblock

Yellowblock

Lightgreenblock

Orangeblock

Grayblock

Contentblock

(farblos)

Auflistungen

Text

- Auflistung
  Umbruch
- Auflistung
  - Auflistung
  - Auflistung

Bei Frames ohne Titel wird die Kopfzeile nicht angezeigt, und der freie Platz kann für Inhalte genutzt werden.

Bei Frames mit Option `[plain]` werden weder Kopf- noch Fußzeile angezeigt.

Beispielinhalt

Bei Frames mit Option `[t]` werden die Inhalte nicht vertikal zentriert, sondern an der Oberkante begonnen.

Beispielinhalt: Literatur

Farbpalette

kit-green100 kit-green90 kit-green80 kit-green70 kit-green60 kit-green50 kit-green40 kit-green30 kit-green25 kit-green20 kit-green15 kit-green10 kit-green5

kit-blue100 kit-blue90 kit-blue80 kit-blue70 kit-blue60 kit-blue50 kit-blue40 kit-blue30 kit-blue25 kit-blue20 kit-blue15 kit-blue10 kit-blue5

kit-red100 kit-red90 kit-red80 kit-red70 kit-red60 kit-red50 kit-red40 kit-red30 kit-red25 kit-red20 kit-red15 kit-red10 kit-red5

kit-gray100 kit-gray90 kit-gray80 kit-gray70 kit-gray60 kit-gray50 kit-gray40 kit-gray30 kit-gray25 kit-gray20 kit-gray15 kit-gray10 kit-gray5

kit-orange100 kit-orange90 kit-orange80 kit-orange70 kit-orange60 kit-orange50 kit-orange40 kit-orange30 kit-orange25 kit-orange20 kit-orange15 kit-orange10 kit-orange5

kit-lightgreen100 kit-lightgreen90 kit-lightgreen80 kit-lightgreen70 kit-lightgreen60 kit-lightgreen50 kit-lightgreen40 kit-lightgreen30 kit-lightgreen25 kit-lightgreen20 kit-lightgreen15
kit-lightgreen10 kit-lightgreen5

kit-brown100 kit-brown90 kit-brown80 kit-brown70 kit-brown60 kit-brown50 kit-brown40 kit-brown30 kit-brown25 kit-brown20 kit-brown15 kit-brown10 kit-brown5

kit-purple100 kit-purple90 kit-purple80 kit-purple70 kit-purple60 kit-purple50 kit-purple40 kit-purple30 kit-purple25 kit-purple20 kit-purple15 kit-purple10 kit-purple5

kit-cyan100 kit-cyan90 kit-cyan80 kit-cyan70 kit-cyan60 kit-cyan50 kit-cyan40 kit-cyan30 kit-cyan25 kit-cyan20 kit-cyan15 kit-cyan10 kit-cyan5