Formal Methods in Software Engineering

Dynamic Logic

Bernhard Beckert



UNIVERSITÄT KOBLENZ-LANDAU

Logical basis

Typed first-order predicate logic

(Types, variables, terms, formulas, ...)

Logical basis

```
Typed first-order predicate logic
```

(Types, variables, terms, formulas, ...)

Assumption for examples

The signature contains a type *Nat* and appropriate symbols:

- function symbols 0, s, +, * (terms $s(0), s(s(0)), \ldots$ written as $1, 2, \ldots$)
- predicate symbols \doteq , <, \leq , >, \geq

NOTE: This is a "convenient assumption" not a definition

WHILE: A Simple Programming Language

Programs

• Assignments:X := tX: variable, t: term• Test:if B then α else β fi
 α , β : programsB: quantifier-free formula,
 α, β : programs• Loop:while B do α od
 α : programsB: quantifier-free formula,
 α : programs• Composition: $\alpha; \beta$ α, β programs

WHILE is computationally complete

Compute the square of X and store it in Y

Y := X * X

Compute the square of *X* and store it in *Y*

Y := X * X

If X is positive then add one else subtract one

 $\underline{if} X > 0 \underline{then} X := X + 1 \underline{else} X := X - 1 \underline{fi}$

```
Compute the square of X (the complicated way)
Making use of: n^2 = 1 + 3 + 5 + \dots + (2n - 1)
       I := 0;
       Y := 0;
       while I < X do
                                             \alphasquare
         Y := Y + 2 * I + 1;
         I := I + 1
       od
```

Russian multiplication

Z := 0;<u>while</u> \neg ($B \doteq 0$) <u>do</u> \underline{if} ((B/2) * 2 \doteq B) \underline{then} A := 2 * A;B := B/2else Z := Z + A;A := 2 * A;B := B/2fi

<u>od</u>

 α_{mult}

Given

A (fixed) first-order structure \mathcal{A} interpreting the function and predicate symbols in the signature

State

$$s = (\mathcal{A}, \beta)$$
 where

 β a variable assignment (i.e. function interpreting the variables)

State update

$$s[X/e] = (\mathcal{A}, \beta[X/e])$$

with

$$\beta[X/e](Y) = \begin{cases} e & \text{if } Y = X \\ \beta(Y) & \text{otherwise} \end{cases}$$

WHILE: Operational Semantics

Define the relation $s[[\alpha]]s'$ as follows

- s[[X := t]]s' iff s' = s[X/s(t)]
- S [[if B then α else β fi]]s' iff
 s ⊨ B and s [[α]]s' or s ⊨ ¬ B and s [[β]]s'
- $s[[\underline{while} \ B \ \underline{do} \ \alpha \ \underline{od}]]s'$ iff there are states $s = s_0, \ldots, s_t = s'$ s.t. $s_i \models B$ for $0 \le i \le t-1$ and $s_t \models \neg B$ and $s_0[[\alpha]]s_1, s_1[[\alpha]]s_2, \ldots, s_{t-1}[[\alpha]]s_t$
- $s[[\alpha; \beta]]s'$ iff there is a state s'' such that $s[[\alpha]]s''$ and $s''[[\beta]]s'$

$[\![\alpha]\!]$ is a partial function

Programs

- X := t (atomic program)
- $\alpha; \beta$ (sequential composition)
- $\alpha \cup \beta$ (non-deterministic choice)
- α^* (non-deterministic iteration, *n* times for some $n \ge 0$)
- F? (test)

remains in initial state if *F* is true, does not terminate if *F* is false

Restriction to deterministic programs

Non-deterministic program constructors may only be used in

 $\underline{\mathbf{if}} \ B \ \underline{\mathbf{then}} \ \alpha \ \underline{\mathbf{else}} \ \beta \ \underline{\mathbf{fi}} \qquad \equiv \qquad (B?; \ \alpha) \cup ((\neg B)?; \ \beta)$

<u>while</u> $B \operatorname{\underline{do}} \alpha \operatorname{\underline{od}} \equiv (B?; \alpha)^*; (\neg B)?$

Logic for expressing properties

Full first-order logic (usually with arithmetic)

Logic for expressing properties

Full first-order logic (usually with arithmetic)

Partial correctness assertion (Hoare formula)

 $\{P\} \alpha \{Q\}$

Meaning:

If α is started in a state satisfying P and terminates, then its final state satisfies Q

Formally:

 $\{P\} \alpha \{Q\}$ is valid iff for all states s, s', if $s \models P$ and $s[[\alpha]]s'$, then $s' \models Q$

$$\{ true \} X := X + 1 \{ X > 1 \}$$

 $\{even(X)\} X := X + 2 \{even(X)\}$

where $even(X) \equiv \exists Z (X \doteq 2 * Z)$

{**true**} α_{square} {Y = X * X}

```
Z := 0;
assert X \doteq A \land Y \doteq B;
while \neg (B \doteq 0) do
     assert A * B + Z \doteq X * Y;
     if ((B/2) * 2 \doteq B) then
          A := 2 * A;
          B := B/2
     else
          Z := Z + A;
          A := 2 * A;
          B := B/2
     fi
od
assert B \doteq 0
assert Z \doteq X * Y
```

Note

X, *Y* are "external" variables

The idea of dynamic logic

- Annotated programs use formulas within programs
- Dynamic Logic uses programs within formulas
- Instead of "assert *F*" after program segment α , write: $[\alpha]F$

The idea of dynamic logic

- Annotated programs use formulas within programs
- Dynamic Logic uses programs within formulas
- Instead of "assert F" after program segment α , write: $[\alpha]F$

A multi-modal logic

- the states are the possible worlds
- ullet two modalities [lpha] and $\langle lpha
 angle$ for each program lpha
- state s' is α -reachable from state s iff $s[\![\alpha]\!]s'$

Semantics

• $[\alpha]F$ true in a state *s* iff *F* is true in all states that are α -reachable from *s*

(partial correctness)

• $\langle \alpha \rangle F$ true in a state *s* iff *F* is true in some state that is α -reachable from *s*

(total correctness)

A formula is valid iff it is valid in all states

Example formulas (validity depends on α,β) $(\langle \alpha \rangle X \doteq Y) \leftrightarrow (\langle \beta \rangle X \doteq Y)$ $\exists X \langle \alpha \rangle$ true **Example formulas** (validity depends on α,β) $(\langle \alpha \rangle X \doteq Y) \leftrightarrow (\langle \beta \rangle X \doteq Y)$ $\exists X \langle \alpha \rangle$ true

Valid formulas

 $[X := 1] X \doteq 1$

[<u>while</u> true <u>do</u> X := X <u>od</u>] false

 $\langle \alpha^* \rangle F \to (F \lor \langle \alpha^* \rangle (\neg F \land \langle \alpha \rangle F))$

Example formulas (validity depends on α,β) $(\langle \alpha \rangle X \doteq Y) \leftrightarrow (\langle \beta \rangle X \doteq Y)$ $\exists X \langle \alpha \rangle$ true

Valid formulas

 $[X := 1] X \doteq 1$

[<u>while</u> true <u>do</u> X := X <u>od</u>] false

 $\langle \alpha^* \rangle F \to (F \lor \langle \alpha^* \rangle (\neg F \land \langle \alpha \rangle F))$

Multiplication example

 $\forall A, B, X, Y, Z(X \doteq A \land Y \doteq B \rightarrow [\alpha_{mult}] Z \doteq X * Y)$

Hoare formulas

 $\{P\} \alpha \{Q\}$ the same as $P \rightarrow [\alpha] Q$

Hoare formulas

 $\{P\} \alpha \{Q\}$ the same as $P \rightarrow [\alpha]Q$

Duality of the modal operators

 $[\alpha] P \leftrightarrow \neg \langle \alpha \rangle \neg P$

Assumption: X does not occur in π

- $(\exists X \langle \pi \rangle F) \quad \leftrightarrow \quad (\langle \pi \rangle \exists XF)$
- $(\forall X [\pi]F) \quad \leftrightarrow \quad ([\pi] \forall XF)$
- $(\exists X [\pi]F) \rightarrow ([\pi] \exists XF)$
- $([\pi] \exists X F) \quad \rightarrow \quad (\exists X [\pi] F)$

provided π is deterministic

 $(\langle \pi \rangle \forall X F) \longrightarrow (\forall X \langle \pi \rangle F)$

 $(\forall X \langle \pi \rangle F) \rightarrow (\langle \pi \rangle \forall X F)$

- provided π is deterministic
- $(\langle \pi \rangle (F \wedge G)) \longrightarrow ((\langle \pi \rangle F) \wedge \langle \pi \rangle G)$
- $((\langle \pi \rangle F) \land \langle \pi \rangle G) \quad \rightarrow \quad (\langle \pi \rangle (F \land G))$

provided π is deterministic

Sequent

 $\Gamma \rightarrow \Delta$

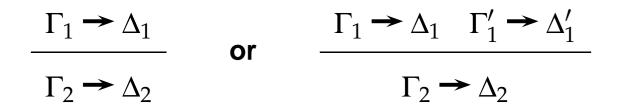
Meaning

 $\wedge \Gamma$ logically implies $\vee \Delta$

(for all variable assignments, i.e.,

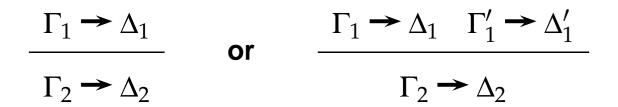
free variables in the sequent are implicitly universally quantified)

Form of sequent rules



(rules can also have more than two premisses)

Form of sequent rules



(rules can also have more than two premisses)

Meaning

The conclusion is true in a state whenever all premisses are true in that state

In particular:

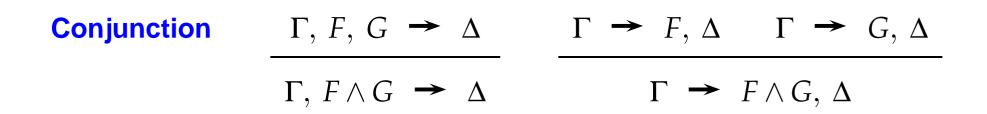
The conclusion is valid whenever all premisses are valid



 Axioms
 $F, \Gamma \rightarrow F, \Delta$ false, $\Gamma \rightarrow \Delta$ $\Gamma \rightarrow$ true, Δ

 Negation
 $\Gamma \rightarrow F, \Delta$ $\Gamma, F \rightarrow \Delta$
 $\Gamma, \neg F \rightarrow \Delta$ $\Gamma, F \rightarrow \Delta$

Axioms $F, \Gamma \rightarrow F, \Delta$ false, $\Gamma \rightarrow \Delta$ $\Gamma \rightarrow$ true, Δ $\Gamma, F \rightarrow \Delta$ $\Gamma \rightarrow F, \Delta$ **Negation** $\Gamma, \neg F \rightarrow \Delta$ $\Gamma \rightarrow \neg F, \Delta$ $\Gamma \rightarrow F, \Delta \qquad \Gamma, G \rightarrow \Delta$ $\Gamma, F \rightarrow G, \Delta$ Implication $\Gamma \rightarrow F \rightarrow G, \Delta$ $\Gamma, F \to G \twoheadrightarrow \Delta$



Conjunction	$\Gamma, F, G \rightarrow \Delta$	$\Gamma \twoheadrightarrow F, \Delta \qquad \Gamma \twoheadrightarrow G, \Delta$
	$\Gamma, F \wedge G \rightarrow \Delta$	$\Gamma \rightarrow F \wedge G, \Delta$
Disjunction	$\Gamma, F \rightarrow \Delta \Gamma, G$	$\rightarrow \Delta$ $\Gamma \rightarrow F, G, \Delta$
	$\Gamma, F \lor G \rightarrow \Delta$	$\Gamma \twoheadrightarrow F \lor G, \Delta$

Universal quantification

$$\Gamma, \forall XF, F\{X \leftarrow t\} \rightarrow \Delta$$

 $\Gamma, \forall XF \rightarrow \Delta$

t an arbitrary term, $\{X \leftarrow t\}$ admissible for *F*

$$\Gamma \twoheadrightarrow F\{X \leftarrow Z\}, \Delta$$

$$\Gamma \rightarrow \forall XF, \Delta$$

Z a **new** variable

Universal quantification

$$\Gamma, \forall XF, F\{X \leftarrow t\} \rightarrow \Delta$$

 $\Gamma, \forall XF \rightarrow \Delta$

t an arbitrary term, $\{X \leftarrow t\}$ admissible for F

$$\Gamma \twoheadrightarrow F\{X \leftarrow Z\}, \Delta$$

$$\Gamma \rightarrow \forall XF, \Delta$$

Z a **new** variable

Existential quantification

$$\Gamma \rightarrow \exists XF, F\{X \leftarrow t\}, \Delta \qquad \Gamma, F\{X \leftarrow Z\} \rightarrow$$

$$\Gamma \rightarrow \exists XF, \Delta \qquad \Gamma, \exists XF \rightarrow \Delta$$

$$t \text{ an arbitrary term,} \qquad Z \text{ a new variable}$$

$$\{X \leftarrow t\} \text{ admissible for } F$$

Λ

6	p(V), p(U)	$axiom \rightarrow p$	$p(U), \forall Y p(Y)$
		1	-right
5	p(V)	I	$p(U), p(U) \rightarrow \forall Y p(Y)$
		ex-ri	ght
4	p(V)	$\rightarrow p$	$p(U), \exists X(p(X) \rightarrow \forall Y p(Y))$
		all-ri	ght
3	p(V)	\rightarrow	$\forall Y p(Y), \exists X (p(X) \rightarrow \forall Y p(Y))$
		impl	-right
2		\rightarrow p	$p(V) \rightarrow \forall Y p(Y), \exists X (p(X) \rightarrow \forall Y p(Y))$
		ex-ri	ght
1		\rightarrow	$X(p(X) \to \forall Y p(Y))$
			Formal Methods in Softwar

Admissibility of Substitutions

Motivation

We want to have that

$$\begin{array}{rccc} \forall XF & \to & F\sigma \\ F\sigma & \to & \exists XF \end{array}$$

is valid for all formulas F and substitutions σ

Definition

A substitution

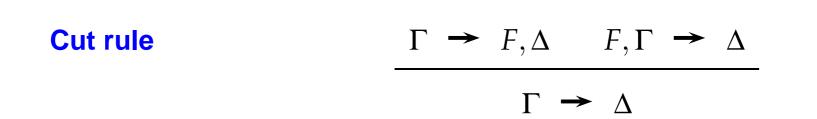
 $\{X \leftarrow t\}$

is admissible for a formula *F* iff

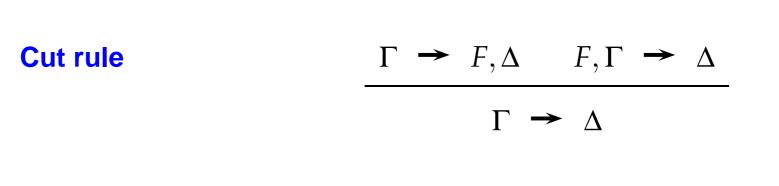
there is no variable Y such that

- Y occurs in t
- there is a quantification $\forall Y$ or $\exists Y$ in F
- \bullet there is a free occurrencence of X in the scope of that quantification

Sequent Calculus for Dynamic Logic



Sequent Calculus for Dynamic Logic



Equality rules

$$\Gamma \twoheadrightarrow t \doteq t, \Delta$$

$$s \doteq t, \Gamma\{s \leftarrow t\} \twoheadrightarrow \Delta\{s \leftarrow t\} \qquad t \doteq s, \Gamma\{s \leftarrow t\} \twoheadrightarrow \Delta\{s \leftarrow t\}$$

$$s \doteq t, \Gamma \twoheadrightarrow \Delta \qquad t \doteq s, \Gamma \twoheadrightarrow \Delta$$

Oracle for first-order logic

 $\Gamma \rightarrow \Delta$

if no programs occur in Γ, Δ and $\mathcal{A} \models \bigwedge \Gamma \rightarrow \bigvee \Delta$

Only of theoretical use! Not computable!

Composition rule

$$\Gamma \rightarrow [\alpha][\beta]F, \Delta$$

$$\Gamma \rightarrow [\alpha; \beta] F, \Delta$$

Composition rule

$$\Gamma \rightarrow [\alpha][\beta]F, \Delta$$
$$\Gamma \rightarrow [\alpha; \beta]F, \Delta$$

Assignment rule

$$\frac{\Gamma\{X \leftarrow X'\}, \ X \doteq t\{X \leftarrow X'\} \twoheadrightarrow F, \ \Delta\{X \leftarrow X'\}}{\Gamma \twoheadrightarrow [X := t]F, \ \Delta} \qquad X' \text{ a new variable}$$

Example:

$$even(X'), X \doteq X' + 2 \rightarrow even(X)$$

$$even(X) \rightarrow [X := X + 2] even(X)$$

A Sequent Calculus for Dynamic Logic

Conditional rule

$$\Gamma, B \twoheadrightarrow [\alpha] F, \Delta \qquad \Gamma, \neg B \twoheadrightarrow [\beta] F, \Delta$$

 $\Gamma \rightarrow [\underline{if} B \underline{then} \alpha \underline{else} \beta \underline{fi}]F, \Delta$

To prove

[while B do body dod] F

find an (arbitrary) formula *Inv* such that

- **1.** Inv is true before execution of the loop
- **2.** $Inv \land B \rightarrow [body]Inv$ is true
- **3.** $Inv \land \neg B \rightarrow F$ is true

Note

Inv is a loop invariant

Loop rule

$$\Gamma \rightarrow Inv, \Delta \qquad Inv, B \rightarrow [\alpha] Inv \qquad Inv, \neg B \rightarrow F$$
$$\Gamma \rightarrow [\underline{while} B \underline{do} \alpha \underline{od}] F, \Delta$$

$$\rightarrow \quad [\alpha_{square}] Y \doteq X * X$$

B: I < X

→ $[I := 0; Y := 0; \text{ while } B \text{ do } \alpha \text{ od}] Y \doteq X * X$ → $[\alpha_{square}] Y \doteq X * X$

B: I < X

 $\rightarrow [I := 0] [Y := 0] [while B do \alpha od] Y \doteq X * X$ $\rightarrow [I := 0; Y := 0; while B do \alpha od] Y \doteq X * X$ $\rightarrow [\alpha_{square}] Y \doteq X * X$

B: I < X

- $I \doteq 0 \quad \rightarrow \quad [Y := 0] [\underline{\text{while}} \ B \ \underline{\text{do}} \ \alpha \ \underline{\text{od}}] \ Y \doteq X * X$
 - $\rightarrow \qquad [I:=0][Y:=0][\underline{\text{while}} \ B \ \underline{\text{do}} \ \alpha \ \underline{\text{od}}] \ Y \doteq X * X$
 - → $[I := 0; Y := 0; \text{ while } B \text{ do } \alpha \text{ od}] Y \doteq X * X$
 - $\rightarrow \quad [\alpha_{square}] Y \doteq X * X$

- B: I < X
- α : Y := Y + 2 * I + 1; I := I + 1

- $I \doteq 0, Y \doteq 0 \rightarrow$ [while $B \operatorname{do} \alpha \operatorname{od} Y \doteq X * X$
 - $I \doteq 0 \quad \rightarrow \quad [Y := 0] \, [\underline{\text{while}} \, B \, \underline{\text{do}} \, \alpha \, \underline{\text{od}}] \, Y \doteq X * X$
 - → [I := 0] [Y := 0] [while $B \text{ do } \alpha \text{ od}] Y \doteq X * X$
 - → $[I := 0; Y := 0; \text{ while } B \text{ do } \alpha \text{ od}] Y \doteq X * X$
 - $\rightarrow \quad [\alpha_{square}] Y \doteq X * X$
- B: I < X
- α : Y := Y + 2 * I + 1; I := I + 1

Invariant *Inv*: $I \leq X \land Y \doteq I * I$

 $I \doteq 0, Y \doteq 0 \rightarrow Inv \quad Inv, B \rightarrow [\alpha] Inv \quad Inv, \neg B \rightarrow Y \doteq X * X$

 $I \doteq 0, Y \doteq 0 \rightarrow$ [while $B \operatorname{\underline{do}} \alpha \operatorname{\underline{od}} Y \doteq X * X$

$$I \doteq 0 \longrightarrow [Y := 0] [$$
while $B \text{ do } \alpha \text{ od}] Y \doteq X * X$

- $\rightarrow \qquad [I:=0] [Y:=0] [\underline{\text{while}} \ B \ \underline{\text{do}} \ \alpha \ \underline{\text{od}}] \ Y \doteq X * X$
- → $[I := 0; Y := 0; while B do \alpha do] Y \doteq X * X$

 $\rightarrow \quad [\alpha_{square}] Y \doteq X * X$

B: I < X



Left branch (pre-condition implies invariant)

$$I \doteq 0, Y \doteq 0 \quad \longrightarrow \quad I \leq X \land Y \doteq I * I$$

Left branch (pre-condition implies invariant)

$$I \doteq 0, \ Y \doteq 0 \quad \longrightarrow \quad 0 \le X \ \land \ Y \doteq 0 * 0$$
$$I \doteq 0, \ Y \doteq 0 \quad \longrightarrow \quad I \le X \ \land \ Y \doteq I * I$$

Example

Left branch (pre-condition implies invariant)

$$I \doteq 0, \ Y \doteq 0 \implies 0 \le X \qquad I \doteq 0, \ Y \doteq 0 \implies Y \doteq 0 * 0$$
$$I \doteq 0, \ Y \doteq 0, \ Y \doteq 0 \implies 0 \le X \land Y \doteq 0 * 0$$
$$I \doteq 0, \ Y \doteq 0 \implies 0 \implies I \le X \land Y \doteq I * I$$



Inv, $B \rightarrow [\alpha]$ Inv



$$I \le X, Y \doteq I * I, I < X \rightarrow [Y := Y + 2 * I + 1; I := I + 1] Inv$$

Inv, $B \rightarrow [\alpha] Inv$



$$I \leq X, \ Y \doteq I * I, \ I < X \quad \rightarrow \quad [Y := Y + 2 * I + 1] [I := I + 1] Inv$$
$$I \leq X, \ Y \doteq I * I, \ I < X \quad \rightarrow \quad [Y := Y + 2 * I + 1; \ I := I + 1] Inv$$
$$Inv, \ B \quad \rightarrow \quad [\alpha] Inv$$

$$I \leq X, \ \mathbf{Y'} \doteq I * I, \ I < X, \ \mathbf{Y} \coloneqq \mathbf{Y'} + 2 * I + 1 \qquad \rightarrow \qquad [I \coloneqq I + 1] Inv$$
$$I \leq X, \ \mathbf{Y} \doteq I * I, \ I < X \qquad \rightarrow \qquad [\mathbf{Y} \coloneqq \mathbf{Y} + 2 * I + 1] [I \coloneqq I + 1] Inv$$
$$I \leq X, \ \mathbf{Y} \doteq I * I, \ I < X \qquad \rightarrow \qquad [\mathbf{Y} \coloneqq \mathbf{Y} + 2 * I + 1; \ I \coloneqq I + 1] Inv$$
$$Inv, \ B \qquad \rightarrow \qquad [\alpha] Inv$$

$$I' \leq X, Y' \doteq I' * I', I' < X, Y \doteq Y' + 2 * I' + 1, I \doteq I' + 1 \rightarrow Inv$$

$$I \leq X, Y' \doteq I * I, I < X, Y \coloneqq Y' + 2 * I + 1 \rightarrow [I \coloneqq I + 1] Inv$$

$$I \leq X, Y \doteq I * I, I < X \rightarrow [Y \coloneqq Y + 2 * I + 1] [I \coloneqq I + 1] Inv$$

$$I \leq X, Y \doteq I * I, I < X \rightarrow [Y \coloneqq Y + 2 * I + 1] [I \coloneqq I + 1] Inv$$

$$Inv, B \rightarrow [\alpha] Inv$$

$$I' < X, I \doteq I' + 1 \implies I \le X$$

$$Y' \doteq I' * I', Y \doteq Y' + 2 * I' + 1, I \doteq I' + 1 \implies Y \doteq I * I$$

$$I' \le X, Y' \doteq I' * I', I' < X, Y \doteq Y' + 2 * I' + 1, I \doteq I' + 1 \implies Inv$$

$$I \le X, Y' \doteq I * I, I < X, Y \coloneqq Y' + 2 * I + 1 \implies [I \coloneqq I + 1] Inv$$

$$I \le X, Y \doteq I * I, I < X \implies [Y \coloneqq Y + 2 * I + 1] [I \coloneqq I + 1] Inv$$

$$I \le X, Y \doteq I * I, I < X \implies [Y \coloneqq Y + 2 * I + 1] [I \coloneqq I + 1] Inv$$

$$I \le X, Y \doteq I * I, I < X \implies [Y \coloneqq Y + 2 * I + 1] [I \coloneqq I + 1] Inv$$

$$I \le X, Y \doteq I * I, I < X \implies [Y \coloneqq Y + 2 * I + 1] [I \coloneqq I + 1] Inv$$

$$I \le X, Y \doteq I * I, I < X \implies [Y \coloneqq Y + 2 * I + 1] [I \coloneqq I + 1] Inv$$

Right branch (invariant and negated loop condition imply post-condition)

 $Inv \wedge \neg B \rightarrow Q$

Right branch (invariant and negated loop condition imply post-condition)

$$I \le X, \ Y \doteq I * I, \ \neg (I < X) \quad \twoheadrightarrow \quad Y \doteq X * X$$
$$Inv \land \neg B \quad \twoheadrightarrow \quad Q$$

Right branch (invariant and negated loop condition imply post-condition)

$$I \le X, \ Y \doteq I * I, \ \neg (I < X) \quad \longrightarrow \quad I \doteq X, \ Y \doteq X * X$$
$$I \le X, \ Y \doteq I * I, \ \neg (I < X) \quad \longrightarrow \quad Y \doteq X * X$$
$$Inv \land \neg B \quad \longrightarrow \quad Q$$

Example

Right branch (invariant and negated loop condition imply post-condition)

$$I \leq X, \ Y \doteq I * I, \ \neg (I < X) \quad \rightarrow \quad I \doteq X \quad Y \doteq I * I$$
$$I \leq X, \ Y \doteq I * I, \ \neg (I < X) \quad \rightarrow \quad I \doteq X, \ Y \doteq X * X$$
$$I \leq X, \ Y \doteq I * I, \ \neg (I < X) \quad \rightarrow \quad Y \doteq X * X$$
$$I \leq X, \ Y \doteq I * I, \ \neg (I < X) \quad \rightarrow \quad Y \doteq X * X$$

$X \doteq A, Y \doteq B \longrightarrow [\alpha_{mult}] Z \doteq X * Y$

 $\begin{array}{lll} X \doteq A, \ Y \doteq B & \twoheadrightarrow & [Z := 0; \ \alpha_{while}] \ Z \doteq X * Y \\ X \doteq A, \ Y \doteq B & \twoheadrightarrow & [\alpha_{mult}] \ Z \doteq X * Y \end{array}$

$$\begin{aligned} X \doteq A, \ Y \doteq B, \ Z \doteq 0 & \longrightarrow & [\alpha_{while}] \ Z \doteq X * Y \\ X \doteq A, \ Y \doteq B & \longrightarrow & [Z := 0; \ \alpha_{while}] \ Z \doteq X * Y \\ X \doteq A, \ Y \doteq B & \longrightarrow & [\alpha_{mult}] \ Z \doteq X * Y \end{aligned}$$

Invariant Inv: $A * B + Z \doteq X * Y$

$$\begin{aligned} X \doteq A, \ Y \doteq B, \ Z \doteq 0 & \longrightarrow \quad Inv \\ Inv, \ \neg B \doteq 0 & \longrightarrow \quad [\alpha_{body}] Inv \\ Inv, \ B \doteq 0 & \longrightarrow \quad Z \doteq X \\ X \doteq A, \ Y \doteq B, \ Z \doteq 0 & \longrightarrow \quad [\alpha_{while}] \ Z \doteq X * Y \\ X \doteq A, \ Y \doteq B & \longrightarrow \quad [Z := 0; \ \alpha_{while}] \ Z \doteq X * Y \\ X \doteq A, \ Y \doteq B & \longrightarrow \quad [\alpha_{mult}] \ Z \doteq X * Y \end{aligned}$$

Left branch (pre-condition implies invariant)

 $X \doteq A, Y \doteq B, Z \doteq 0 \implies A * B + Z \doteq X * Y$

Left branch (pre-condition implies invariant)

 $X \doteq A, Y \doteq B, Z \doteq 0 \implies A * B + Z \doteq X * Y$

$$A * B + Z \doteq X * Y, \ \neg B \doteq 0 \quad \longrightarrow \quad [\alpha_{body}]A * B + Z \doteq X * Y$$

Left branch (pre-condition implies invariant)

 $X \doteq A, Y \doteq B, Z \doteq 0 \rightarrow A * B + Z \doteq X * Y$

Middle branch (invariant is indeed invariant)

 $A * B + Z \doteq X * Y, \neg B \doteq 0 \rightarrow [\alpha_{body}]A * B + Z \doteq X * Y$

Right branch (invariant and negated loop condition imply post-condition)

 $A * B + Z \doteq X * Y, B \doteq 0 \rightarrow Z \doteq X * Y$

Purpose

- Needed to prove first-order theorems on natural numbers (oracle not available in practice)
- **•** Handling loops in $\langle \cdot \rangle$ modality

Purpose

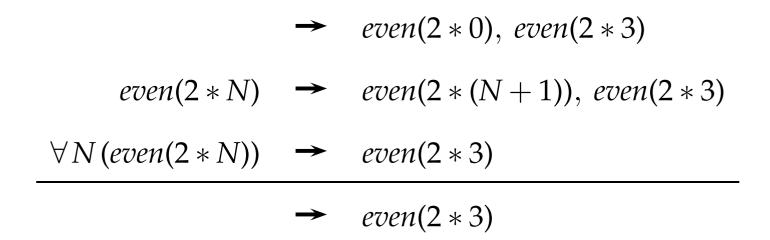
- Needed to prove first-order theorems on natural numbers (oracle not available in practice)
- Handling loops in $\langle \cdot \rangle$ modality

$\Gamma \twoheadrightarrow F\{N \leftarrow 0\}, \Delta \qquad \Gamma, F \twoheadrightarrow F\{N \leftarrow N+1\}, \Delta \qquad \Gamma, \forall NF \twoheadrightarrow \Delta$

$\Gamma \rightarrow \Delta$

N not occurring in Γ , Δ

N not occurring in any program in F



Rule

$\Gamma, \neg B \twoheadrightarrow F, \Delta \qquad \Gamma, B \twoheadrightarrow \langle \alpha \rangle \langle \underline{\text{while}} B \underline{\text{do}} \alpha \underline{\text{od}} \rangle F, \Delta$ $\Gamma \twoheadrightarrow \langle \underline{\text{while}} B \underline{\text{do}} \alpha \underline{\text{od}} \rangle F, \Delta$

Rule

$$\Gamma, \neg B \rightarrow F, \Delta \qquad \Gamma, B \rightarrow \langle \alpha \rangle \langle \underline{\text{while }} B \underline{\text{do}} \alpha \underline{\text{od}} \rangle F, \Delta$$
$$\Gamma \rightarrow \langle \underline{\text{while }} B \underline{\text{do}} \alpha \underline{\text{od}} \rangle F, \Delta$$

Note

Only useful

- in connection with induction rule, or
- if number of loop iterations has a (small) known upper bound

Loop Unwind Rule / Induction Rule: Example

Proof goal

$$\rightarrow \quad \langle \underline{\mathsf{while}} \ I > 0 \ \underline{\mathsf{do}} \ I := I - 1 \ \underline{\mathsf{od}} \rangle \ I \doteq 0$$

Induction hypothesis

$$F(N) \equiv \forall I (I \le N \rightarrow \langle \underline{\text{while}} I > 0 \underline{\text{do}} I := I - 1 \underline{\text{od}} \rangle I \doteq 0)$$

Previous definition of admissibility is not sufficient if formulas contain programs

$$F \equiv J \doteq K \rightarrow [I := 0] (J \doteq K)$$
 valid

Previous definition of admissibility is not sufficient if formulas contain programs

$$F \equiv J \doteq K \rightarrow [I := 0] (J \doteq K)$$
valid
$$F\{I \leftarrow J\} \equiv J \doteq K \rightarrow [J := 0] (J \doteq K)$$
not valid

Previous definition of admissibility is not sufficient if formulas contain programs

$$F \equiv J \doteq K \rightarrow [I := 0] (J \doteq K) \quad \text{valid}$$

$$F\{I \leftarrow J\} \equiv J \doteq K \rightarrow [J := 0] (J \doteq K) \quad \text{not valid}$$

$$F\{J \leftarrow I\} \equiv I \doteq K \rightarrow [I := 0] (I \doteq K) \quad \text{not valid}$$

Previous definition of admissibility is not sufficient if formulas contain programs

$$F \equiv J \doteq K \rightarrow [I := 0] (J \doteq K) \quad \text{valid}$$

$$F\{I \leftarrow J\} \equiv J \doteq K \rightarrow [J := 0] (J \doteq K) \quad \text{not valid}$$

$$F\{J \leftarrow I\} \equiv I \doteq K \rightarrow [I := 0] (I \doteq K) \quad \text{not valid}$$

$$F\{I \leftarrow 1\} \equiv J \doteq K \rightarrow [1 := 0] (J \doteq K) \quad \text{not a formula}$$

Revised definition

A substitution $\{X \leftarrow t\}$ is admissible for a formula *F* iff

1. t = X, or

- **2.** t is a variable not occurring in F, or
- 3. there is no variable Y in t such that a free occurrence of X in F is in the scope of
 - (a) a quantification $\forall Y \text{ or } \exists Y$, or
 - (b) a modality containing an assignment of the form Y := s