Hoare Calculation and its Application

Robert Lang

TUM

March 2008 - Saint Petersburg - JASS 2008

A first example

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

A first example

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

How can we prove, that for $x, y \in \mathbb{N}_0$:

$$\texttt{result(x,y)} = x + y$$

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on *x*:

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on x:

$x = 0$

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on x:

$x = 0 \Rightarrow$ `result(0,y)` $\underbrace{=}_{\texttt{x==0}} y$, and $y = y + x$

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on x:

$x = 0 \Rightarrow \texttt{result(0,y)} \underbrace{=}_{\texttt{x==0}} y$, and $y = y + x$ ✓

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on x:

$x = 0 \Rightarrow \texttt{result(0,y)} \underbrace{=}_{\texttt{x==0}} y$, and $y = y + x$ ✓

Let the assumption be proved for some $x \in \mathbb{N}_0$ and all $y \in \mathbb{N}_0$.

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on x:

$x = 0 \Rightarrow \texttt{result(0,y)} \underbrace{=}_{\texttt{x==0}} y$, and $y = y + x$ ✓

Let the assumption be proved for some $x \in \mathbb{N}_0$ and all $y \in \mathbb{N}_0$.

$\Rightarrow \texttt{result(x+1,y)} \underbrace{=}_{\texttt{else}} \texttt{result(x,y+1)}$

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on x:

$x = 0 \Rightarrow \texttt{result(0,y)} \underbrace{=}_{\texttt{x==0}} y$, and $y = y + x$ ✓

Let the assumption be proved for some $x \in \mathbb{N}_0$ and all $y \in \mathbb{N}_0$.

$\Rightarrow \texttt{result(x+1,y)} \underbrace{=}_{\texttt{else}} \texttt{result(x,y+1)}$

$$\underbrace{=}_{\text{induction hypothese}} x + (y + 1)$$

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on x:

$x = 0 \Rightarrow \texttt{result(0,y)} \underbrace{=}_{\texttt{x==0}} y$, and $y = y + x$ ✓

Let the assumption be proved for some $x \in \mathbb{N}_0$ and all $y \in \mathbb{N}_0$.

$$\Rightarrow \texttt{result(x+1,y)} \underbrace{=}_{\texttt{else}} \texttt{result(x,y+1)}$$

$$\underbrace{=}_{\text{induction hypothese}} x + (y+1)$$

$$\Rightarrow \texttt{result(x+1,y)} = x + (y+1) = (x+1) + y$$

```
function result(x,y)
  if x == 0
    return (y);
  else
    result(x-1,y+1);
  end
```

Proof of the assertion by induction on x:

$x = 0 \Rightarrow \texttt{result(0,y)} \underbrace{=}_{\texttt{x==0}} y$, and $y = y + x$ ✓

Let the assumption be proved for some $x \in \mathbb{N}_0$ and all $y \in \mathbb{N}_0$.

$$\Rightarrow \texttt{result(x+1,y)} \underbrace{=}_{\texttt{else}} \texttt{result(x,y+1)}$$

$$\underbrace{=}_{\text{induction hypothese}} x + (y+1)$$

$$\Rightarrow \texttt{result(x+1,y)} = x + (y+1) = (x+1) + y \qquad \square$$

A second example

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

A second example

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

How can we prove, that for $x, y \in \mathbb{N}_0$:

$$\texttt{result\_2(x,y)} = x + y$$

A second example

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

How can we prove, that for $x, y \in \mathbb{N}_0$:

$$\texttt{result\_2(x,y)} = x + y$$

As easy as in the first example?

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

Try to prove the assertion by induction on x:

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

Try to prove the assertion by induction on x:

$x = 0$

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

Try to prove the assertion by induction on x:

$x = 0 \Rightarrow \texttt{result\_2(0,y)} = y$ and $y = y + x$

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

Try to prove the assertion by induction on x:

$x = 0 \Rightarrow \texttt{result\_2(0,y)} = y$ and $y = y + x$ ✓

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

Try to prove the assertion by induction on x:

$x = 0 \Rightarrow$ `result_2(0,y)` $= y$ and $y = y + x$ ✓

Let the assumption be proved for some $x \in \mathbb{N}_0$ and all $y \in \mathbb{N}_0$.

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

Try to prove the assertion by induction on x:

$x = 0 \Rightarrow$ `result_2(0,y)` $= y$ and $y = y + x$ ✓

Let the assumption be proved for some $x \in \mathbb{N}_0$ and all $y \in \mathbb{N}_0$.

`result_2(x+1,y)`

```
function result_2(x,y)
  while x > 0
    x = x-1;
    y = y+1;
  end
  return y;
```

<u>Try to prove the assertion by induction on x:</u>

$x = 0 \Rightarrow$ `result_2(0,y)` $= y$ and $y = y + x$ ✓

Let the assumption be proved for some $x \in \mathbb{N}_0$ and all $y \in \mathbb{N}_0$.
`result_2(x+1,y)` $= \ldots$ It doesn't work!

What is the problem?

What is the problem?

- There's no recursive run of `result_2`.

What is the problem?

- There's no recursive run of `result_2`.
- Number of `while`-loop-iterations depends on x.

What is the problem?

- There's no recursive run of `result_2`.
- Number of `while`-loop-iterations depends on x.
- Values of x, y are changing during running time.

What is the problem?

- There's no recursive run of `result_2`.
- Number of `while`-loop-iterations depends on x.
- Values of x, y are changing during running time.

$\Rightarrow$ Mathematical methods of proof won't last!

What is the problem?

- There's no recursive run of `result_2`.
- Number of `while`-loop-iterations depends on x.
- Values of x, y are changing during running time.

$\Rightarrow$ Mathematical methods of proof won't last!

$\Rightarrow$ We need new tools!

Challenges

Let `P` be a given program. We want to prove, that

Challenges

Let `P` be a given program. We want to prove, that

1. `P` terminates for all valid inputs.

Challenges

Let `P` be a given program. We want to prove, that

1. `P` terminates for all valid inputs.
2. `P` works for a given domain in that way it is built for.

Challenges

Let `P` be a given program. We want to prove, that

1. `P` terminates for all valid inputs.
2. `P` works for a given domain in that way it is built for.

- Both tasks are as hard as the Halting Problem.

Challenges

Let `P` be a given program. We want to prove, that

1. `P` terminates for all valid inputs.
2. `P` works for a given domain in that way it is built for.

- Both tasks are as hard as the Halting Problem.
- You must prove them for every single `P`.

Challenges

Let `P` be a given program. We want to prove, that

1. `P` terminates for all valid inputs.
2. `P` works for a given domain in that way it is built for.

- Both tasks are as hard as the Halting Problem.
- You must prove them for every single `P`.

In the following algorithms the termination is assumed.

Challenges

Let `P` be a given program. We want to prove, that

1. `P` terminates for all valid inputs.
2. `P` works for a given domain in that way it is built for.

- Both tasks are as hard as the Halting Problem.
- You must prove them for every single `P`.

In the following algorithms the termination is assumed.
⇒ We just meet challenge 2 using Hoare Calculation . . .

C.A.R. Hoare

C.A.R. Hoare

Sir Charles Antony Richard Hoare
(*11. January 1934 Colombo, Sri Lanka)

Source: Wikipedia, the free encyclopedia

C.A.R. Hoare

Sir Charles Antony Richard Hoare
(*11. January 1934 Colombo, Sri Lanka)

Source: Wikipedia, the free encyclopedia

> *"I conclude that there are two ways of constructing a software design:*

C.A.R. Hoare

Sir Charles Antony Richard Hoare
(*11. January 1934 Colombo, Sri Lanka)

Source: Wikipedia, the free encyclopedia

> *"I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies*

C.A.R. Hoare

Sir Charles Antony Richard Hoare
(*11. January 1934 Colombo, Sri Lanka)

Source: Wikipedia, the free encyclopedia

> *"I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies."*

Hoare-Triple

$$\{\mathtt{P}\}\ \mathbf{S}\ \{\mathtt{Q}\}$$

Hoare-Triple

$$\{\texttt{P}\}\ \mathbf{S}\ \{\texttt{Q}\}$$

- `P`, `Q` predicates with values `true` or `false`

Hoare-Triple

$$\{\mathtt{P}\}\ \mathbf{S}\ \{\mathtt{Q}\}$$

- `P`, `Q` predicates with values `true` or `false`
- **S** statement, a program with correct syntax

Hoare-Triple

$$\{\mathtt{P}\}\ \mathbf{S}\ \{\mathtt{Q}\}$$

- `P`, `Q` predicates with values `true` or `false`
- **S** statement, a program with correct syntax

Hoare-Triples are binary expressions with values `true` or `false`.

Hoare-Triple

$$\{\mathtt{P}\}\ \mathbf{S}\ \{\mathtt{Q}\}$$

- $\mathtt{P}$, $\mathtt{Q}$ predicates with values `true` or `false`
- **S** statement, a program with correct syntax

Hoare-Triples are binary expressions with values `true` or `false`.

$\{\mathtt{P}\}\ \mathbf{S}\ \{\mathtt{Q}\} = \mathtt{true} :\Leftrightarrow$

Hoare-Triple

$$\{\texttt{P}\}\ \mathbf{S}\ \{\texttt{Q}\}$$

- `P`, `Q` predicates with values `true` or `false`
- **S** statement, a program with correct syntax

Hoare-Triples are binary expressions with values `true` or `false`.

$\{\texttt{P}\}\ \mathbf{S}\ \{\texttt{Q}\} = \texttt{true} :\Leftrightarrow$
If the predicate $\{\texttt{P}\}$ is true immediately before execution of **S**, then immediately **S** has terminated, the predicate $\{\texttt{Q}\}$ is true.

Hoare-Triple

$$\{\texttt{P}\}\ \mathbf{S}\ \{\texttt{Q}\}$$

- `P`, `Q` predicates with values `true` or `false`
- **S** statement, a program with correct syntax

Hoare-Triples are binary expressions with values `true` or `false`.

$\{\texttt{P}\}\ \mathbf{S}\ \{\texttt{Q}\} = \texttt{true} :\Leftrightarrow$
If the predicate $\{\texttt{P}\}$ is true immediately before execution of **S**, then immediately **S** has terminated, the predicate $\{\texttt{Q}\}$ is true.

Notation: $\dfrac{X}{Y}$

Hoare-Triple

$$\{\mathtt{P}\}\ \mathbf{S}\ \{\mathtt{Q}\}$$

- `P`, `Q` predicates with values `true` or `false`
- **S** statement, a program with correct syntax

Hoare-Triples are binary expressions with values `true` or `false`.

$\{\mathtt{P}\}\ \mathbf{S}\ \{\mathtt{Q}\} = \mathtt{true} :\Leftrightarrow$
If the predicate $\{\mathtt{P}\}$ is true immediately before execution of **S**, then immediately **S** has terminated, the predicate $\{\mathtt{Q}\}$ is true.

Notation: $\frac{X}{Y} :\Leftrightarrow X \Rightarrow Y$

Hoare Rule 1: Skip-Axiom

$$\frac{\texttt{true}}{\{\texttt{A}\}\ \textbf{skip}\ \{\texttt{A}\}}$$

Hoare Rule 1: Skip-Axiom

$$\frac{\mathtt{true}}{\{\mathtt{A}\}\ \textbf{skip}\ \{\mathtt{A}\}}$$

skip means the program with no commands.

Hoare Rule 2: Axiom of Assignment

$$\frac{\texttt{true}}{\{\texttt{A}_{\beta/x}\}\ x{:=}\beta\ \{\texttt{A}\}}$$

Hoare Rule 2: Axiom of Assignment

$$\frac{\texttt{true}}{\{\texttt{A}_{\beta/x}\}\ x{:=}\beta\ \{\texttt{A}\}}$$

$\texttt{A}_{\beta/x}$ is predicate $\texttt{A}$, but x instead of β.

Hoare Rule 3: Rule of Composition

$$\frac{\{\mathtt{A}\}\ \mathbf{S1}\ \{\mathtt{B}\} \wedge \{\mathtt{B}\}\ \mathbf{S2}\ \{\mathtt{C}\}}{\{\mathtt{A}\}\ \mathbf{S1},\mathbf{S2}\ \{\mathtt{C}\}}$$

Hoare Rule 4: Rule of Conditional Branching

$$\frac{\{\mathtt{A} \wedge \mathtt{B}\}\ \mathbf{S1}\ \{\mathtt{Q}\} \wedge \{\mathtt{A} \wedge \neg\mathtt{B}\}\ \mathbf{S2}\ \{\mathtt{Q}\}}{\{\mathtt{A}\}\ \mathbf{if}\ \mathtt{B}\ \mathbf{then}\ \mathbf{S1}\ \mathbf{else}\ \mathbf{S2}\ \mathbf{end\ if}\ \{\mathtt{Q}\}}$$

Hoare Rule 5: Rule of Iteration

$$\frac{\{\mathtt{I} \wedge \mathtt{B}\}\ \textbf{S}\ \{\mathtt{I}\}}{\{\mathtt{I}\}\ \textbf{while}\ \mathtt{B}\ \textbf{loop}\ \textbf{S}\ \textbf{end loop}\ \{\mathtt{I} \wedge \neg\mathtt{B}\}}$$

Hoare Rule 5: Rule of Iteration

$$\frac{\{\mathtt{I} \wedge \mathtt{B}\}\ \mathbf{S}\ \{\mathtt{I}\}}{\{\mathtt{I}\}\ \textbf{while}\ \mathtt{B}\ \textbf{loop}\ \mathbf{S}\ \textbf{end loop}\ \{\mathtt{I} \wedge \neg\mathtt{B}\}}$$

Such an `I` is called loop-invariant.

Hoare Rule 6: Rule of Consequence

$$\frac{\mathtt{A} \Rightarrow \mathtt{A'} \quad \wedge \{\mathtt{A'}\}\ \mathtt{S}\ \{\mathtt{B'}\} \wedge \quad \mathtt{B'} \Rightarrow \mathtt{B}}{\{\mathtt{A}\}\ \mathbf{S}\ \{\mathtt{B}\}}$$

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)
```

```
function result_2(x,y)
```

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)
```

```
function result_2(x,y)
```

$\{P\colon x \geq 0 \wedge y \geq 0,\ r := x + y\}$

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)


  while x > 0
```

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
    {I}
  while x > 0
```

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)


  while x > 0

    x = x-1;
    y = y+1;

  end
```

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
    {I}
   while x > 0
    {I ∧ B}
    x = x-1;
    y = y+1;
    {I}
  end
```

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)


  while x > 0

    x = x-1;
    y = y+1;

  end
```

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
    {I}
   while x > 0
    {I $\wedge$ B}
     x = x-1;
     y = y+1;
    {I}
   end
    {I $\wedge \neg$ B}  (*Rule of Iteration*)
```

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)


  while x > 0


    x = x-1;
    y = y+1;

  end
```

function result_2(x,y)

$\{P\colon x \geq 0 \wedge y \geq 0,\ r := x + y\}$

$\{I\}$

while x > 0

$\{I \wedge B\}$

x = x-1;

y = y+1;

$\{I\}$

end

$\{I \wedge \neg B\}$ (*Rule of Iteration*)

$\{Q\colon y = r\}$

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)


  while x > 0


    x = x-1;
    y = y+1;

  end


  return y;
```

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
    {I}
   while x > 0
    {I ∧ B}
    x = x-1;
    y = y+1;
    {I}
   end
    {I ∧ ¬ B}  (Rule of Iteration)
    {Q: $y = r$}
   return y;
```

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)

  while x > 0

    x = x-1;
    y = y+1;

  end

  return y;
```

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0,\ r := x + y$}
    {I}
   while x > 0
    {I ∧ B}
    x = x-1;
    y = y+1;
    {I}
   end
    {I ∧ ¬ B}  (Rule of Iteration)
    {Q: $y = r$}
   return y;
```

- B: $x > 0$ (condition in `while`-loop)

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)


  while x > 0

    x = x-1;
    y = y+1;

  end


  return y;
```

```
function result_2(x,y)
   {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
   {I}
  while x > 0
   {I $\wedge$ B}
   x = x-1;
   y = y+1;
   {I}
  end
   {I $\wedge$ ¬ B}  (Rule of Iteration)
   {Q: $y = r$}
  return y;
```

- B: $x > 0$ (condition in `while`-loop) $\Rightarrow$ ¬ B: $\neg(x > 0)$

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)

  while x > 0

    x = x-1;
    y = y+1;

  end

  return y;
```

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
    {I}
  while x > 0
    {I ∧ B}
    x = x-1;
    y = y+1;
    {I}
  end
    {I ∧ ¬ B}  (Rule of Iteration)
    {Q: $y = r$}
  return y;
```

- B: $x > 0$ (condition in while-loop) ⇒ ¬ B: $\neg(x > 0)$
- loop-invariant:

Proof of `result_2(x,y)` using Hoare

```
function result_2(x,y)


  while x > 0

    x = x-1;
    y = y+1;

  end


  return y;
```

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0,\ r := x + y$}
    {I}
  while x > 0
    {$I \wedge B$}
    x = x-1;
    y = y+1;
    {I}
  end
    {$I \wedge \neg B$}  (Rule of Iteration)
    {Q: $y = r$}
  return y;
```

- B: $x > 0$ (condition in `while`-loop) $\Rightarrow \neg$ B: $\neg(x > 0)$
- loop-invariant: I: $r = x + y$

```
while x > 0
```

```
while x > 0
```

$\{\mathrm{I}\colon r = x + y \wedge \mathrm{B}\colon x > 0\}$

```
while x > 0
   {I: $r = x + y \wedge$ B: $x > 0$}
   {Item 1}
   x = x-1;
   {Item 2}
   y = y+1;
```

```
while x > 0
   {I: $r = x + y \wedge$ B: $x > 0$}
   {Item 1}
   x = x-1;
   {Item 2}
   y = y+1;
   {I: $r = x + y,\ x \geq 0$}
```

```
while x > 0
  {I: $r = x + y \land$ B: $x > 0$}
  {Item 1}
  x = x-1;
  {Item 2}
  y = y+1;
  {I: $r = x + y,\ x \geq 0$}
```

- Rule of Assign.:

```
while x > 0
  {I: $r = x + y \wedge$ B: $x > 0$}
  {Item 1}
  x = x-1;
  {Item 2}
  y = y+1;
  {I: $r = x + y$, $x \geq 0$}
```

- Rule of Assign.: Item 2: $\{x + (y + 1),\ x \geq 0\}$

```
while x > 0
   {I: $r = x + y \wedge$ B: $x > 0$}
   {Item 1}
   x = x-1;
   {Item 2}
   y = y+1;
   {I: $r = x + y$, $x \geq 0$}
```

- Rule of Assign.: Item 2: $\{x + (y + 1),\ x \geq 0\}$
- Assign.:

```
while x > 0
    {I: $r = x + y \wedge$ B: $x > 0$}
    {Item 1}
    x = x-1;
    {Item 2}
    y = y+1;
    {I: $r = x + y$, $x \geq 0$}
```

- Rule of Assign.: Item 2: $\{x + (y + 1), x \geq 0\}$
- Assign.: Item 1: $\{(x - 1) + (y + 1), x - 1 \geq 0\}$

```
while x > 0
   {I: $r = x + y \wedge$ B: $x > 0$}
   {Item 1}
   x = x-1;
   {Item 2}
   y = y+1;
   {I: $r = x + y,\ x \geq 0$}
```

- Rule of Assign.: Item 2: $\{x + (y + 1),\ x \geq 0\}$
- Assign.: Item 1: $\{(x - 1) + (y + 1),\ x - 1 \geq 0\}$
- In fact:

```
while x > 0
  {I: $r = x + y \wedge$ B: $x > 0$}
  {Item 1}
  x = x-1;
  {Item 2}
  y = y+1;
  {I: $r = x + y,\ x \geq 0$}
```

- Rule of Assign.: Item 2: $\{x + (y + 1),\ x \geq 0\}$
- Assign.: Item 1: $\{(x - 1) + (y + 1),\ x - 1 \geq 0\}$
- In fact: $\{(x - 1) + (y + 1),\ x - 1 \geq 0\}$

```
while x > 0
  {I: $r = x + y \wedge$ B: $x > 0$}
  {Item 1}
  x = x-1;
  {Item 2}
  y = y+1;
  {I: $r = x + y,\ x \geq 0$}
```

- Rule of Assign.: Item 2: $\{x + (y + 1),\ x \geq 0\}$
- Assign.: Item 1: $\{(x - 1) + (y + 1),\ x - 1 \geq 0\}$
- In fact: $\{(x - 1) + (y + 1),\ x - 1 \geq 0\} \triangleq \{$I: $x + y \wedge$ B: $x > 0\}$

```
while x > 0
   {I: $r = x + y \wedge$ B: $x > 0$}
   {Item 1}
   x = x-1;
   {Item 2}
   y = y+1;
   {I: $r = x + y,\ x \geq 0$}
```

- Rule of Assign.: Item 2: $\{x + (y + 1),\ x \geq 0\}$
- Assign.: Item 1: $\{(x - 1) + (y + 1),\ x - 1 \geq 0\}$
- In fact: $\{(x - 1) + (y + 1),\ x - 1 \geq 0\} \triangleq \{$I: $x + y \wedge$ B: $x > 0\}$ because $x - 1 \geq 0 \Leftrightarrow x > 0$ for integer x.

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
    {I: $r = x + y$}
    while x > 0
        {I: $r = x + y \wedge$ B: $x > 0 \;\; y \geq 0$}
        x = x-1;
        y = y+1;
        {I: $r = x + y$}
    end
```

```
function result_2(x,y)
  {P: $x \geq 0 \wedge y \geq 0,\ r := x + y$}
  {I: $r = x + y$}
  while x > 0
    {I: $r = x + y \wedge$ B: $x > 0\ \ y \geq 0$}
    x = x-1;
    y = y+1;
    {I: $r = x + y$}
  end
  {I: $r = x + y \wedge \neg$ B: $\neg(x > 0)$}
```

```
function result_2(x,y)
  {P: $x \geq 0 \wedge y \geq 0,\ r := x + y$}
  {I: $r = x + y$}
  while x > 0
    {I: $r = x + y \wedge$ B: $x > 0 \quad y \geq 0$}
    x = x-1;
    y = y+1;
    {I: $r = x + y$}
  end
  {I: $r = x + y \wedge \neg$ B: $\neg(x > 0)$}
  {Q: $y = r$}
```

```
function result_2(x,y)
  {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
  {I: $r = x + y$}
  while x > 0
    {I: $r = x + y \wedge$ B: $x > 0 \;\; y \geq 0$}
    x = x-1;
    y = y+1;
    {I: $r = x + y$}
  end
  {I: $r = x + y \wedge \neg$ B: $\neg(x > 0)$}
  {Q: $y = r$}
  return y;
```

```
function result_2(x,y)
  $\{P: x \geq 0 \wedge y \geq 0,\ r := x + y\}$
  $\{I: r = x + y\}$
  while x > 0
    $\{I: r = x + y \wedge B: x > 0 \quad y \geq 0\}$
    x = x-1;
    y = y+1;
    $\{I: r = x + y\}$
  end
  $\{I: r = x + y \wedge \neg B: \neg(x > 0)\}$
  $\{Q: y = r\}$
  return y;
```

P: $x \geq 0 \wedge y \geq 0$

```
function result_2(x,y)
  {$P$: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
  {$I$: $r = x + y$}
  while x > 0
    {$I$: $r = x + y \wedge$ $B$: $x > 0 \;\; y \geq 0$}
    x = x-1;
    y = y+1;
    {$I$: $r = x + y$}
  end
  {$I$: $r = x + y \wedge \neg$ $B$: $\neg(x > 0)$}
  {$Q$: $y = r$}
  return y;
```

P: $x \geq 0 \wedge y \geq 0$ and $\neg(x > 0)$

```
function result_2(x,y)
  {$P$: $x \geq 0 \wedge y \geq 0,\ r := x + y$}
  {$I$: $r = x + y$}
  while x > 0
    {$I$: $r = x + y \wedge B$: $x > 0\ \ y \geq 0$}
    x = x-1;
    y = y+1;
    {$I$: $r = x + y$}
  end
  {$I$: $r = x + y \wedge \neg B$: $\neg(x > 0)$}
  {$Q$: $y = r$}
  return y;
```

P: $x \geq 0 \wedge y \geq 0$ and $\neg(x > 0) \Rightarrow x = 0$

```
function result_2(x,y)
  $\{P\colon x \geq 0 \wedge y \geq 0,\ r := x + y\}$
  $\{I\colon r = x + y\}$
  while x > 0
    $\{I\colon r = x + y \wedge B\colon x > 0 \;\; y \geq 0\}$
    x = x-1;
    y = y+1;
    $\{I\colon r = x + y\}$
  end
  $\{I\colon r = x + y \wedge \neg B\colon \neg(x > 0)\}$
  $\{Q\colon y = r\}$
  return y;
```

P: $x \geq 0 \wedge y \geq 0$ and $\neg(x > 0) \Rightarrow x = 0$

$\Rightarrow$ Q: $y =$

```
function result_2(x,y)
  {P: $x \geq 0 \wedge y \geq 0$, $r := x + y$}
  {I: $r = x + y$}
  while x > 0
    {I: $r = x + y \wedge$ B: $x > 0 \;\; y \geq 0$}
    x = x-1;
    y = y+1;
    {I: $r = x + y$}
  end
  {I: $r = x + y \wedge \neg$ B: $\neg(x > 0)$}
  {Q: $y = r$}
  return y;
```

P: $x \geq 0 \wedge y \geq 0$ and $\neg(x > 0) \Rightarrow x = 0$

$\Rightarrow$ Q: $y = y + x$

```
function result_2(x,y)
  $\{P\text{: } x \geq 0 \wedge y \geq 0,\ r := x + y\}$
  $\{I\text{: } r = x + y\}$
  while x > 0
    $\{I\text{: } r = x + y \wedge B\text{: } x > 0 \quad y \geq 0\}$
    x = x-1;
    y = y+1;
    $\{I\text{: } r = x + y\}$
  end
  $\{I\text{: } r = x + y \wedge \neg\ B\text{: } \neg(x > 0)\}$
  $\{Q\text{: } y = r\}$
  return y;
```

P: $x \geq 0 \wedge y \geq 0$ and $\neg(x > 0) \Rightarrow x = 0$
$\Rightarrow$ Q: $y = y + x = r$ (I: $r = x + y$ loop-invariant)

```
function result_2(x,y)
    {P: $x \geq 0 \wedge y \geq 0,\ r := x + y$}
    {I: $r = x + y$}
    while x > 0
        {I: $r = x + y \wedge$ B: $x > 0 \;\; y \geq 0$}
        x = x-1;
        y = y+1;
        {I: $r = x + y$}
    end
    {I: $r = x + y \wedge \neg$ B: $\neg(x > 0)$}
    {Q: $y = r$}
    return y;
```

P: $x \geq 0 \wedge y \geq 0$ and $\neg(x > 0) \Rightarrow x = 0$

$\Rightarrow$ Q: $y = y + x = r$ (I: $r = x + y$ loop-invariant) □

Numerical Quadrature

Numerical Quadrature

Let $f : [a, b] \to \mathbb{R}$ be sufficiently smooth (e.g. $f \in C^2$).

Numerical Quadrature

Let $f : [a, b] \longrightarrow \mathbb{R}$ be sufficiently smooth (e.g. $f \in C^2$).

The functional of the definite integral is given by

Numerical Quadrature

Let $f : [a, b] \longrightarrow \mathbb{R}$ be sufficiently smooth (e.g. $f \in C^2$).

The functional of the definite integral is given by

$$F(f, a, b) := \int_a^b f(x)\,dx$$

Numerical Quadrature

Let $f : [a, b] \longrightarrow \mathbb{R}$ be sufficiently smooth (e.g. $f \in C^2$).

The functional of the definite integral is given by

$$F(f, a, b) := \int_a^b f(x)\, dx$$

Numerical Quadrature means:

Numerical Quadrature

Let $f : [a, b] \longrightarrow \mathbb{R}$ be sufficiently smooth (e.g. $f \in C^2$).

The functional of the definite integral is given by

$$F(f, a, b) := \int_a^b f(x)\, dx$$

Numerical Quadrature means:
Calculate an approximation for the numerical value of $F(f, a, b)$.

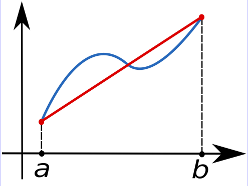
The Trapezodial-Rule

Approximation with linear function:

The Trapezodial-Rule

Approximation with linear function:

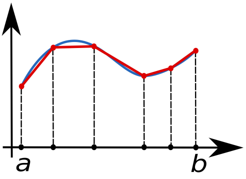
$$F \approx T := (b - a) \cdot \frac{f(a) + f(b)}{2}$$



Dividing $[a, b]$ into smaller, equidistant intervals:

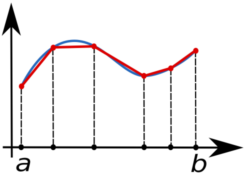
Dividing $[a, b]$ into smaller, equidistant intervals: ⇒ piecewise linear functions

$$F \approx TS := \frac{b-a}{n+1} \cdot \left[\frac{f(a)}{2} + \sum_{k=1}^{n} f(x_k) + \frac{f(b)}{2} \right]$$



Dividing $[a, b]$ into smaller, equidistant intervals: ⇒ piecewise linear functions

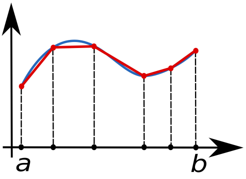
$$F \approx TS := \frac{b-a}{n+1} \cdot \left[\frac{f(a)}{2} + \sum_{k=1}^{n} f(x_k) + \frac{f(b)}{2} \right]$$



In the picture: $n = 4$

Dividing $[a, b]$ into smaller, equidistant intervals: ⇒ piecewise linear functions

$$F \approx TS := \frac{b-a}{n+1} \cdot \left[\frac{f(a)}{2} + \sum_{k=1}^{n} f(x_k) + \frac{f(b)}{2} \right]$$

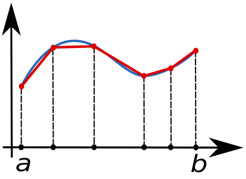


In the picture: $n = 4$

The errors $\Delta F = |F - T|$ or $\Delta F = |F - TS|$ depend on the second derivative:

Dividing $[a, b]$ into smaller, equidistant intervals: ⇒ piecewise linear functions

$$F \approx TS := \frac{b-a}{n+1} \cdot \left[\frac{f(a)}{2} + \sum_{k=1}^{n} f(x_k) + \frac{f(b)}{2} \right]$$



In the picture: $n = 4$

The errors $\Delta F = |F - T|$ or $\Delta F = |F - TS|$ depend on the second derivative:

$$\Delta F \leq \frac{(b-a)^3}{12 \cdot n^2} \cdot ||f''||_\infty$$

Hierachical Decomposition

To approximate $F(f, a, b)$ we start with the Trapezoidal-Rule:

Hierachical Decomposition

To approximate $F(f, a, b)$ we start with the Trapezoidal-Rule:

$$F(f, a, b) \approx T(f, a, b) = (b - a) \cdot \frac{f(a) + f(b)}{2}$$

Hierachical Decomposition

To approximate $F(f,a,b)$ we start with the Trapezoidal-Rule:

$$F(f,a,b) \approx T(f,a,b) = (b-a) \cdot \frac{f(a)+f(b)}{2}$$

There is a residuum $S(f,a,b)$ with:

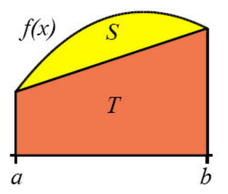
Hierachical Decomposition

To approximate $F(f, a, b)$ we start with the Trapezoidal-Rule:

$$F(f, a, b) \approx T(f, a, b) = (b - a) \cdot \frac{f(a) + f(b)}{2}$$

There is a residuum $S(f, a, b)$ with:

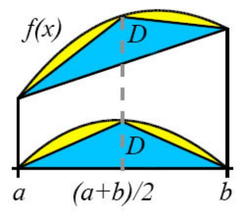
$$F(f, a, b) = T(f, a, b) + S(f, a, b)$$



Now decompose $S(f, a, b)$ into a triangle D with projected heigh

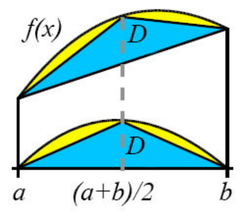
Now decompose $S(f, a, b)$ into a triangle D with projected heigh

$$h = f\left(\frac{a+b}{2}\right) - \frac{f(a)+f(b)}{2}$$



Now decompose $S(f, a, b)$ into a triangle D with projected heigh

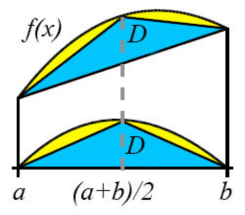
$$h = f\left(\frac{a+b}{2}\right) - \frac{f(a) + f(b)}{2}$$



The area of D is given by:

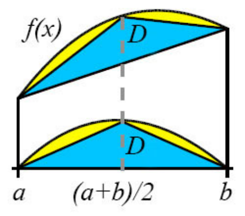
Now decompose $S(f, a, b)$ into a triangle D with projected heigh

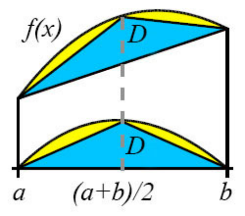
$$h = f\left(\frac{a+b}{2}\right) - \frac{f(a) + f(b)}{2}$$



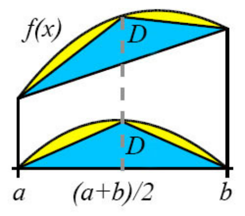
The area of D is given by:

$$D(f, a, b) = \frac{b-a}{2} \cdot h$$


f(x)
D
D
a
(a+b)/2
b



The new residuum can be determined by using this idea recursively:



The new residuum can be determined by using this idea recursively:

$$S(f,a,b) = D(f,a,b) + S(f,a,\frac{a+b}{2}) + S(f,\frac{a+b}{2},b)$$

Approximation via Basis Functions

Approximation via Basis Functions

If $u : [a, b] \to \mathbb{R}$ is an approximation to f, then

Approximation via Basis Functions

If $u : [a, b] \to \mathbb{R}$ is an approximation to f, then

$$F(f, a, b) \approx F(u, a, b)$$

Approximation via Basis Functions

If $u : [a, b] \to \mathbb{R}$ is an approximation to f, then

$$F(f, a, b) \approx F(u, a, b)$$

Let $u(x)$ be a linear combination of basis functions $\Phi_k(x)$:

Approximation via Basis Functions

If $u : [a, b] \rightarrow \mathbb{R}$ is an approximation to f, then

$$F(f, a, b) \approx F(u, a, b)$$

Let $u(x)$ be a linear combination of basis functions $\Phi_k(x)$:

$$u(x) = \sum_{k=1}^{N} \alpha_k \Phi_k(x)$$

Approximation via Basis Functions

If $u : [a, b] \to \mathbb{R}$ is an approximation to f, then

$$F(f, a, b) \approx F(u, a, b)$$

Let $u(x)$ be a linear combination of basis functions $\Phi_k(x)$:

$$u(x) = \sum_{k=1}^{N} \alpha_k \Phi_k(x)$$

Now we can write easily:

Approximation via Basis Functions

If $u : [a, b] \to \mathbb{R}$ is an approximation to f, then

$$F(f, a, b) \approx F(u, a, b)$$

Let $u(x)$ be a linear combination of basis functions $\Phi_k(x)$:

$$u(x) = \sum_{k=1}^{N} \alpha_k \Phi_k(x)$$

Now we can write easily:

$$F(f, a, b) \approx \int_a^b u(x) =$$

Approximation via Basis Functions

If $u : [a, b] \to \mathbb{R}$ is an approximation to f, then

$$F(f, a, b) \approx F(u, a, b)$$

Let $u(x)$ be a linear combination of basis functions $\Phi_k(x)$:

$$u(x) = \sum_{k=1}^{N} \alpha_k \Phi_k(x)$$

Now we can write easily:

$$F(f, a, b) \approx \int_a^b u(x) = \int_a^b \sum_{k=1}^{N} \alpha_k \Phi_k(x) =$$

Approximation via Basis Functions

If $u : [a, b] \to \mathbb{R}$ is an approximation to f, then

$$F(f, a, b) \approx F(u, a, b)$$

Let $u(x)$ be a linear combination of basis functions $\Phi_k(x)$:

$$u(x) = \sum_{k=1}^{N} \alpha_k \Phi_k(x)$$

Now we can write easily:

$$F(f, a, b) \approx \int_a^b u(x) = \int_a^b \sum_{k=1}^{N} \alpha_k \Phi_k(x) = \sum_{k=1}^{N} \alpha_k \int_a^b \Phi_k(x)$$

Define „hat functions“ as basis functions via

Define „hat functions“ as basis functions via

$$\Phi_{n,i} = \Phi\left(\frac{x - x_{n,i}}{h_n}\right)$$

Define „hat functions“ as basis functions via

$$\Phi_{n,i} = \Phi\left(\frac{x - x_{n,i}}{h_n}\right)$$

- $\Phi(x) := \max\{1 - |x|, 0\}$

Define „hat functions" as basis functions via

$$\Phi_{n,i} = \Phi\left(\frac{x - x_{n,i}}{h_n}\right)$$

- $\Phi(x) := \max\{1 - |x|, 0\}$
- mesh size $h_n := 2^{-n}$

Define „hat functions“ as basis functions via

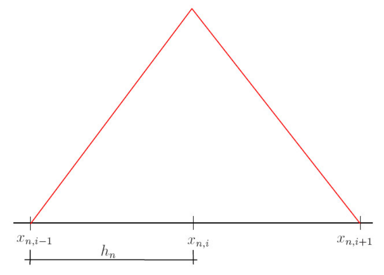
$$\Phi_{n,i} = \Phi\left(\frac{x - x_{n,i}}{h_n}\right)$$

- $\Phi(x) := \max\{1 - |x|, 0\}$
- mesh size $h_n := 2^{-n}$
- grid points $x_{n,i} = i \cdot h_n$

Define „hat functions" as basis functions via

$$\Phi_{n,i} = \Phi\left(\frac{x - x_{n,i}}{h_n}\right)$$

- $\Phi(x) := \max\{1 - |x|, 0\}$
- mesh size $h_n := 2^{-n}$
- grid points $x_{n,i} = i \cdot h_n$



Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

$$\Psi_N := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n\}$$

Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

$$\Psi_N := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n\}$$

is a generator system for V_N (but for $N > 1$ not a basis).

Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

$$\Psi_N := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \le i < 2^n\}$$

is a generator system for V_N (but for $N > 1$ not a basis).
We use the hierachical basis:

Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

$$\Psi_N := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n\}$$

is a generator system for V_N (but for $N > 1$ not a basis).
We use the hierachical basis:

$$\Psi_N^H := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n \mid i \text{ odd}\}$$

Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0, 1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

$$\Psi_N := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n\}$$

is a generator system for V_N (but for $N > 1$ not a basis).
We use the hierachical basis:

$$\Psi_N^H := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n \mid i \text{ odd}\}$$

$$W_n := \mathtt{span}\{\Phi_{n,i} \mid \alpha_{n,i} = 0 \text{ for all even } i\}$$

Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

$$\Psi_N := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n\}$$

is a generator system for V_N (but for $N > 1$ not a basis).
We use the hierachical basis:

$$\Psi_N^H := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n \mid i \text{ odd}\}$$

$$W_n := \mathtt{span}\{\Phi_{n,i} \mid \alpha_{n,i} = 0 \text{ for all even } i\} \Rightarrow V_N = V_{N-1} \oplus W_N$$

Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

$$\Psi_N := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n\}$$

is a generator system for V_N (but for $N > 1$ not a basis).
We use the hierachical basis:

$$\Psi_N^H := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n \mid i \text{ odd}\}$$

$$W_n := \mathtt{span}\{\Phi_{n,i} \mid \alpha_{n,i} = 0 \text{ for all even } i\} \Rightarrow V_N = V_{N-1} \oplus W_N$$

$$\Rightarrow V_N = \oplus_{n=1}^{N} W_n$$

Let V_N be the space of the continuous, on grid h_n piecewise linear functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$. Then

$$\Psi_N := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n\}$$

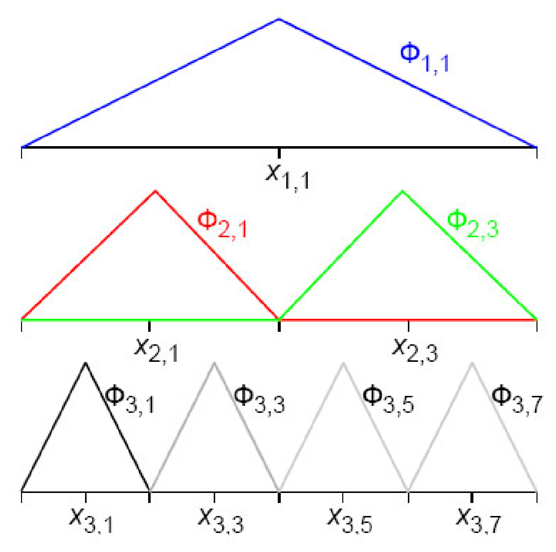
is a generator system for V_N (but for $N > 1$ not a basis).
We use the hierachical basis:

$$\Psi_N^H := \bigcup_{n=1}^{N} \{\Phi_{n,i} : 1 \leq i < 2^n \mid i \text{ odd}\}$$

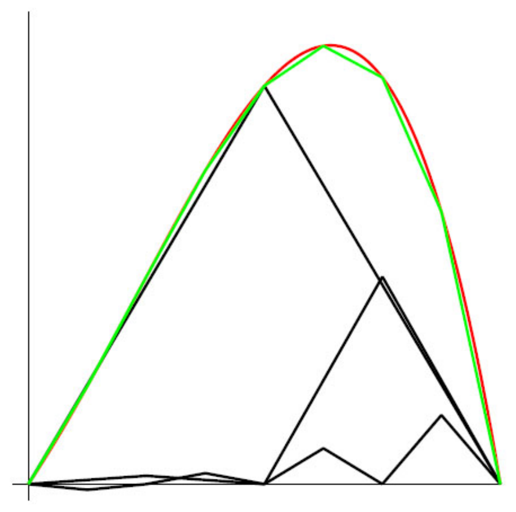
$$W_n := \mathtt{span}\{\Phi_{n,i} \mid \alpha_{n,i} = 0 \text{ for all even } i\} \Rightarrow V_N = V_{N-1} \oplus W_N$$

$$\Rightarrow V_N = \oplus_{n=1}^{N} W_n \text{ (inductive argument with } V_1 = W_1)$$

The hierachical basis for W_1, W_2 and W_3



Approximation



Representation in hierachical basis

Let $v \in V_N$ be a vector:

Representation in hierachical basis

Let $v \in V_N$ be a vector:

$$v(x) = \sum_{n=1}^{N} \sum_{i=1}^{2^n-1} \alpha_{n,i} \Phi_{n,i}(x)$$

Representation in hierachical basis

Let $v \in V_N$ be a vector:

$$v(x) = \sum_{n=1}^{N} \sum_{i=1}^{2^n - 1} \alpha_{n,i} \Phi_{n,i}(x)$$

The program `HierachicalBasis(N)` should convert $v(x)$ into the hierachical basis: $(N > 1)$

Representation in hierachical basis

Let $v \in V_N$ be a vector:

$$v(x) = \sum_{n=1}^{N} \sum_{i=1}^{2^n-1} \alpha_{n,i} \Phi_{n,i}(x)$$

The program `HierachicalBasis(N)` should convert $v(x)$ into the hierachical basis: $(N > 1)$

$$v(x) = \sum_{n=1}^{N} \sum_{i=1}^{2^n-1} \alpha'_{n,i} \Phi_{n,i}(x)$$

with $\alpha'_{n,i} = 0$ for all even i.

Program `HierachicalBasis(N)`

```
function HierachicalBasis(N)
```

Program `HierachicalBasis(N)`

```
function HierachicalBasis(N)
  for n = N-1,...,1 :
```

Program `HierachicalBasis(N)`

```
function HierachicalBasis(N)
  for n = N-1,...,1 :
    for i = 1,...,$2^n - 1$ :
```

Program HierachicalBasis(N)

```
function HierachicalBasis(N)
  for n = N-1,...,1 :
    for i = 1,...,$2^n - 1$ :
      $a_{n+1,2i-1}$ -= $a_{n+1,2i}/2$
      $a_{n+1,2i+1}$ -= $a_{n+1,2i}/2$
```

Program `HierachicalBasis(N)`

```
function HierachicalBasis(N)
  for n = N-1,...,1 :
    for i = $1,\ldots,2^n-1$ :
      $a_{n+1,2i-1}$ - = $a_{n+1,2i}/2$
      $a_{n+1,2i+1}$ - = $a_{n+1,2i}/2$
      $a_{n,i}$ + = $a_{n+1,2i}$
```

Program HierachicalBasis(N)

```
function HierachicalBasis(N)
  for n = N-1,...,1 :
    for i = $1,\ldots,2^n - 1$ :
      $a_{n+1,2i-1}$ -= $a_{n+1,2i}/2$
      $a_{n+1,2i+1}$ -= $a_{n+1,2i}/2$
      $a_{n,i}$ += $a_{n+1,2i}$
      $a_{n+1,2i}$ = $0$
```

Program `HierachicalBasis(N)`

```
function HierachicalBasis(N)
  for n = N-1,...,1 :
    for i = 1,...,$2^n - 1$ :
      $a_{n+1,2i-1}$ -= $a_{n+1,2i}/2$
      $a_{n+1,2i+1}$ -= $a_{n+1,2i}/2$
      $a_{n,i}$ += $a_{n+1,2i}$
      $a_{n+1,2i}$ = 0
```

To prove the correctness of `HierachicalBasis(N)`, the programm must be written in a form Hoare Calculation can handle with:

```
function HierachicalBasis_Hoare(N)
```

```
function HierachicalBasis_Hoare(N)
  n = N-1
  while n $\neq$ 0
```

```
function HierachicalBasis_Hoare(N)
  n = N-1
  while n $\neq$ $0$
    i = 1
    while i $\neq$ $2^n$
```

```
function HierachicalBasis_Hoare(N)
  n = N-1
  while n $\neq 0$
    i = 1
    while i $\neq 2^n$
      $a_{n+1,2i-1} = a_{n+1,2i-1} - a_{n+1,2i}/2$
      $a_{n+1,2i+1} = a_{n+1,2i+1} - a_{n+1,2i}/2$
      $a_{n,i} = a_{n,i} + a_{n+1,2i}$
      $a_{n+1,2i} = 0$
```

```
function HierachicalBasis_Hoare(N)
  n = N-1
  while n $\neq$ 0
    i = 1
    while i $\neq$ $2^n$
      $a_{n+1,2i-1} = a_{n+1,2i-1} - a_{n+1,2i}/2$
      $a_{n+1,2i+1} = a_{n+1,2i+1} - a_{n+1,2i}/2$
      $a_{n,i} = a_{n,i} + a_{n+1,2i}$
      $a_{n+1,2i} = 0$
      i = i+1
    n = n-1
```

Why to use Hierachical Basis

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high):

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon$

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials:

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \texttt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \texttt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

For high-dimensional functions (dimension d) the memory requirements $M \propto N^d$ $(N = \dim V_N)$

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

For high-dimensional functions (dimension d) the memory requirements $M \propto N^d$ ($N = \dim V_N$)
$\Rightarrow$ exponential increasing with d

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

For high-dimensional functions (dimension d) the memory requirements $M \propto N^d$ ($N = \dim V_N$)
$\Rightarrow$ exponential increasing with d
With hierachical basis:

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

For high-dimensional functions (dimension d) the memory requirements $M \propto N^d$ ($N = \dim V_N$)
$\Rightarrow$ exponential increasing with d
With hierachical basis: $M \propto N \cdot (\ln N)^{d-1}$

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \texttt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

For high-dimensional functions (dimension d) the memory requirements $M \propto N^d$ $(N = \dim V_N)$
$\Rightarrow$ exponential increasing with d
With hierachical basis: $M \propto N \cdot (\ln N)^{d-1}$
$\Rightarrow$ New problem:

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \texttt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b - a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

For high-dimensional functions (dimension d) the memory requirements $M \propto N^d$ ($N = \dim V_N$)
$\Rightarrow$ exponential increasing with d
With hierachical basis: $M \propto N \cdot (\ln N)^{d-1}$
$\Rightarrow$ New problem: Program code very complicated!

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

For high-dimensional functions (dimension d) the memory requirements $M \propto N^d$ $(N = \dim V_N)$
$\Rightarrow$ exponential increasing with d
With hierachical basis: $M \propto N \cdot (\ln N)^{d-1}$
$\Rightarrow$ New problem: Program code very complicated! How to be sure, there are no deficiencies?

Why to use Hierachical Basis

- Adaptive stop criterion (via projected high): $\alpha_{n,i} < \epsilon \Rightarrow \mathtt{STOP}$
- The global error can be estimated by $\Delta F \leq \epsilon(b-a)$
- For second-degree polynomials: $S = \frac{4}{3}D$ (Simpsons-Rule)

For high-dimensional functions (dimension d) the memory requirements $M \propto N^d$ ($N = \dim V_N$)
$\Rightarrow$ exponential increasing with d
With hierachical basis: $M \propto N \cdot (\ln N)^{d-1}$
$\Rightarrow$ New problem: Program code very complicated! How to be sure, there are no deficiencies?
$\Rightarrow$ Hoare Calculation!

References

- Michael Gellner: Der Umgang mit dem Hoare-Kalkül zur Programmverikfikation
- Volker Claus: Einführung in die Informatik 2005/06 - Kapitel 7: Semantik von Programmen
- Samuel Kerschbaumer: The Hoare Logic - Providing Numerical Algorithms (2006)
- Peter Heinig: Program Verification using Hoare Logic - An Introduction
- Michael Bader, Stefan Zimmer: Hierarchische Zerlegung (eindimensional)

End of presentation

Thank you for your attention!