# Формални подходи в информатиката

Трифон Трифонов

Разширен семинар по програмиране, 2014/15 г.

# За какво ще говорим?

За математиката в програмирането:

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?
- Как се пишат доказано коректни програми?
- Може ли тези доказателства да стават автоматично?
- Може ли програмата да се проверява за коректност при компилация?

# За какво ще говорим?

За математиката в програмирането:

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?
- Как се пишат доказано коректни програми?
- Може ли тези доказателства да стават автоматично?
- Може ли програмата да се проверява за коректност при компилация?

# За какво ще говорим?

За математиката в програмирането:

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?
- Как се пишат доказано коректни програми?
- Може ли тези доказателства да стават автоматично?
- Може ли програмата да се проверява за коректност при компилация?

# За какво ще говорим?

За математиката в програмирането:

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?
- Как се пишат доказано коректни програми?
- Може ли тези доказателства да стават автоматично?
- Може ли програмата да се проверява за коректност при компилация?

# За какво ще говорим?

За математиката в програмирането:

- Какво означава да **докажем**, че една програма прави това, което очакваме от нея?
- Как се пишат доказано коректни програми?
- Може ли тези доказателства да стават автоматично?
- Може ли програмата да се проверява за коректност при компилация?

# Формална верификация на програми

Нека с $P$ означим програма с входни променливи $\vec{x}$ и изходни променливи $\vec{y} := P(\vec{x})$.

Нека с $A(\vec{x}, \vec{y})$ означим някаква релация между $\vec{x}$ и $\vec{y}$.

**Верификация.** Дадени са $A$ и $P$. Искаме да проверим дали $A$ е в сила за $P$, т.е. дали $A(\vec{x}, P(\vec{x}))$.

Как да направим тази проверка?

# Формална верификация на програми

Нека с $P$ означим програма с входни променливи $\vec{x}$ и изходни променливи $\vec{y} := P(\vec{x})$.

Нека с $A(\vec{x}, \vec{y})$ означим някаква релация между $\vec{x}$ и $\vec{y}$.

**Верификация.** Дадени са $A$ и $P$. Искаме да проверим дали $A$ е в сила за $P$, т.е. дали $A(\vec{x}, P(\vec{x}))$.

Как да направим тази проверка?

# Логика на Хоар

Разглеждаме тройки от вида

{Предусловие} Програма {Следусловие}

Строим дървета с тройки по възлите по следните правила:

$$\{P\}\ ;\ \{P\} \qquad \text{(празен оператор)}$$

$$\{P[x \mapsto E]\}\ x = E;\ \{P\} \qquad \text{(присвояване)}$$

# Логика на Хоар

Разглеждаме тройки от вида

{Предусловие} Програма {Следусловие}

Строим дървета с тройки по възлите по следните правила:

$\{P\}\ ;\ \{P\}$ (празен оператор)

$\{P[x \mapsto E]\}\ x = E;\ \{P\}$ (присвояване)

# Логика на Хоар

Разглеждаме тройки от вида

$$\{\text{Предусловие}\}\ \text{Програма}\ \{\text{Следусловие}\}$$

Строим дървета с тройки по възлите по следните правила:

$$\{P\}\ ;\ \{P\} \qquad \text{(празен оператор)}$$

$$\{P[x \mapsto E]\}\ x = E;\ \{P\} \qquad \text{(присвояване)}$$

# Логика на Хоар (2)

$$\frac{\{P\}\ S\ \{R\} \qquad \{R\}\ T\ \{Q\}}{\{P\}\ S; T\ \{Q\}}$$ (композиция)

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad \{P \wedge \neg B\}\ T\ \{Q\}}{\{P\}\ \textbf{if}\ (B)\ S\ \textbf{else}\ T\ \{Q\}}$$ (условен оператор)

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ (B)\ S\ \{P \wedge \neg B\}}$$ (цикъл)

$P$ — **инварианта** на цикъла

# Логика на Хоар (2)

$$\frac{\{P\}\ S\ \{R\} \qquad \{R\}\ T\ \{Q\}}{\{P\}\ S; T\ \{Q\}}$$ (композиция)

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad \{P \wedge \neg B\}\ T\ \{Q\}}{\{P\}\ \textbf{if}\ (B)\ S\ \textbf{else}\ T\ \{Q\}}$$ (условен оператор)

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ (B)\ S\ \{P \wedge \neg B\}}$$ (цикъл)

$P$ — **инварианта** на цикъла

# Логика на Хоар (2)

$$\frac{\{P\}\ S\ \{R\} \qquad \{R\}\ T\ \{Q\}}{\{P\}\ S; T\ \{Q\}}$$ (композиция)

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad \{P \wedge \neg B\}\ T\ \{Q\}}{\{P\}\ \textbf{if}\ (B)\ S\ \textbf{else}\ T\ \{Q\}}$$ (условен оператор)

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ (B)\ S\ \{P \wedge \neg B\}}$$ (цикъл)

$P$ — **инварианта** на цикъла

# Логика на Хоар (3)

$$\frac{\{P\}\ S\ \{Q\} \qquad R \to P}{\{R\}\ S\ \{Q\}} \quad \text{(усилване)}$$

$$\frac{\{P\}\ S\ \{Q\} \qquad Q \to R}{\{P\}\ S\ \{R\}} \quad \text{(отслабване)}$$

# Логика на Хоар: пример

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

Търсим инварианта $P$ такава, че:

1. $(x = 0 \wedge i = 1) \rightarrow P$
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$

# Логика на Хоар: пример

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

Търсим инварианта $P$ такава, че:

1. $(x = 0 \wedge i = 1) \rightarrow P$
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$

# Логика на Хоар: пример

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

Търсим инварианта $P$ такава, че:

1. $(x = 0 \wedge i = 1) \rightarrow P$
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$

# Логика на Хоар: пример

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

Търсим инварианта $P$ такава, че:

1. $(x = 0 \wedge i = 1) \rightarrow P$
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$

# Логика на Хоар: пример (2)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{n(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$$\boxed{P : x = \frac{n(i-1)}{2}}$$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ?
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

# Логика на Хоар: пример (2)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{n(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{n(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ?
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

# Логика на Хоар: пример (2)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{n(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{n(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ?
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

# Логика на Хоар: пример (2)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{n(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{n(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ?
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

# Логика на Хоар: пример (3)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{i(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$$\boxed{P : x = \frac{i(i-1)}{2}}$$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ✓
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

# Логика на Хоар: пример (3)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{i(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{i(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ✓
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

# Логика на Хоар: пример (3)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{i(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{i(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ✓
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

# Логика на Хоар: пример (3)

$\{x = 0 \wedge i = 1\}$

```
while(i != n) {
```

$\{x = \frac{i(i-1)}{2}\}$

```
  x += i;
  i++;
}
```

$\{x = \frac{n(n-1)}{2}\}$

$\boxed{P : x = \frac{i(i-1)}{2}}$

1. $(x = 0 \wedge i = 1) \rightarrow P$ ✓
2. $\{P \wedge i \neq n\}$ `x += i; i++;` $\{P\}$ ✓
3. $(P \wedge i = n) \rightarrow x = \frac{n(n-1)}{2}$ ✓

# Езици за анотиране на програми

- Java Modeling Language (JML) — Java
- ANSI/ISO C Specification Language (ACSL) — C
- Typed Assembly Language (TAL) — Assembler
- Property Specifiation Language (PSL) — Verilog (хардуер)

# Пример: факториел

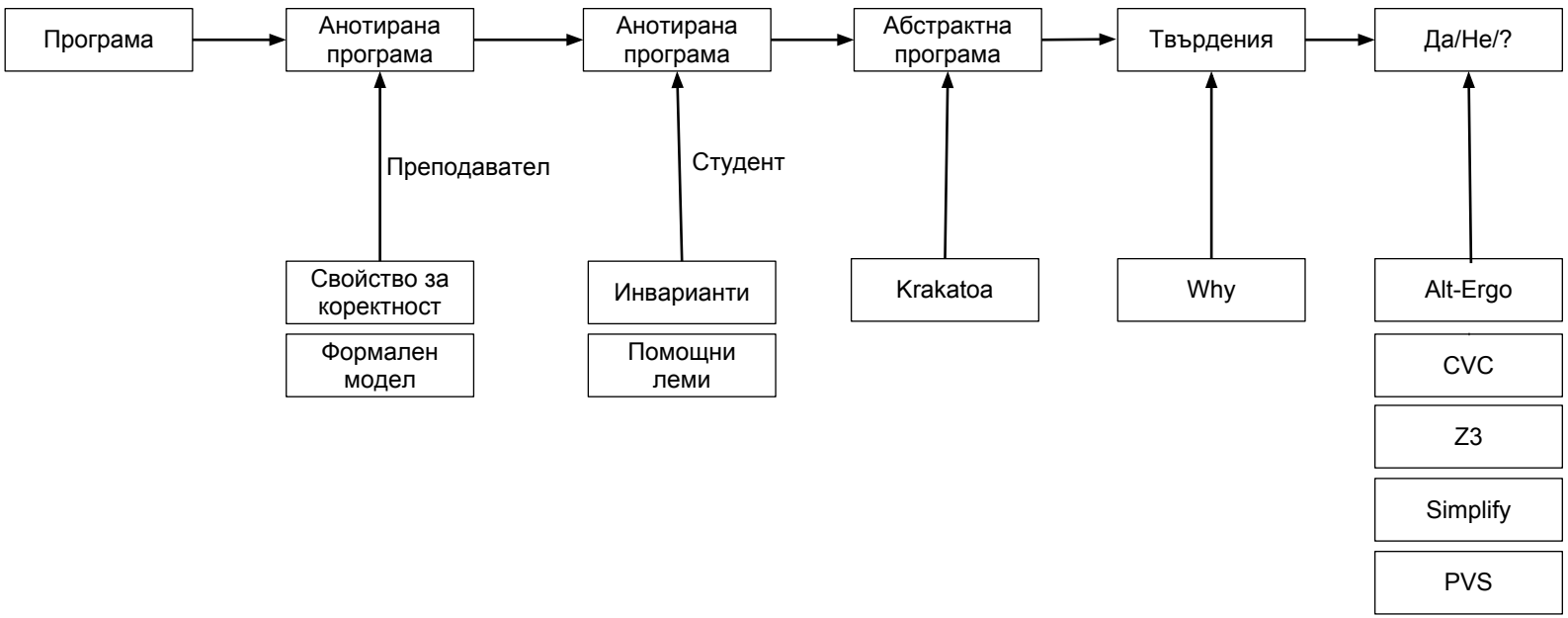
Модел:

```
/*@ inductive isfact(integer x, integer r) {
  @  case isfact0: isfact(0,1);
  @  case isfactn:
  @   \forall integer n r;
  @     n >= 1 && isfact(n-1,r) ==> isfact(n,r*n);
  @ }
  @*/
```

# Пример: факториел (2)

```
/*@ requires x >= 0;
  @ ensures isfact(x, \result);
  @*/
public static int fact(int x) {
    int a = 0, y = 1;
    /*@ loop_invariant 0 <= a <= x && isfact(a,y);
      @ loop_variant x-a;
      @*/
    while (a < x) y = y * ++a;
    return y;
}
```

# Обща схема


Програма
Анотирана програма
Анотирана програма
Абстрактна програма
Твърдения
Да/Не/?
Преподавател
Студент
Свойство за коректност
Формален модел
Инварианти
Помощни леми
Krakatoa
Why
Alt-Ergo
CVC
Z3
Simplify
PVS

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима! Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима!
Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

# Системи за автоматично доказване на теореми

В общия случай задачата за доказване на теореми е нерешима! Но...

- Има системи, за които задачата е решима:
  - съждителна логика, линейна темпорална логика (LTL), аритметика на Presburger
  - за тях формалната верификация се нарича model checking
- Има алгоритми, които работят за широк кръг от теореми
- Има много автоматични доказвачи на теореми, които работят доста добре:
  - Alt-Ergo, Automath, CVC, KeY, PhoX, Princess, PVS, SPASS, TPS, Vampire, ...

Дори има **World Championship for Automated Theorem Proving**

# Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони
- чрез прилагане на стратегии (замяна на константа с променлива, промяна на границите на интервала и др.)
- чрез изследване на предикатите в програмата
- трансформаци на пред- и след-условието

# Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони
- чрез прилагане на стратегии (замяна на константа с променлива, промяна на границите на интервала и др.)
- чрез изследване на предикатите в програмата
- трансформаци на пред- и след-условието

# Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони
- чрез прилагане на стратегии (замяна на константа с променлива, промяна на границите на интервала и др.)
- чрез изследване на предикатите в програмата
- трансформаци на пред- и след-условието

# Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони
- чрез прилагане на стратегии (замяна на константа с променлива, промяна на границите на интервала и др.)
- чрез изследване на предикатите в програмата
- трансформаци на пред- и след-условието

# Системи за автоматично генериране на анотации

- Не винаги е лесно да се сетим за правилните пред- и следусловия
- Най-трудни са инвариантите...
- Какво да правим, ако не можем да измислим инварианти?

Има системи, които могат да ни ги подскажат! Как го правят?

- предварително зададени шаблони
- чрез прилагане на стратегии (замяна на константа с променлива, промяна на границите на интервала и др.)
- чрез изследване на предикатите в програмата
- трансформаци на пред- и след-условието

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

# Системи за автоматична проверка за коректност

Една система за автоматична проверка за коректност обикновено се състои от:

- синтактичен анализатор (парсер) за езика
- синтактичен анализатор за логически свойства
- транслатор на конкретната програма до абстрактна
- генератор на условия за проверка
- генератор на инварианти
- система за автоматична проверка на теореми
- устройство пред екрана :)

Примери: Dafny, Why

# Синтезиране на програми

Нека с $P$ означим програма с входни променливи $\vec{x}$ и изходни променливи $\vec{y} := P(\vec{x})$.

Нека с $A(\vec{x}, \vec{y})$ означим някаква релация между $\vec{x}$ и $\vec{y}$.

**Синтезиране.** Дадено е $A$. Да се намери програма $P$, така че $A$ е в сила за $P$, т.е. $A(\vec{x}, P(\vec{x}))$.

Как да построим програмата $P$?

# Синтезиране на програми

Нека с $P$ означим програма с входни променливи $\vec{x}$ и изходни променливи $\vec{y} := P(\vec{x})$.

Нека с $A(\vec{x}, \vec{y})$ означим някаква релация между $\vec{x}$ и $\vec{y}$.

**Синтезиране.** Дадено е $A$. Да се намери програма $P$, така че $A$ е в сила за $P$, т.е. $A(\vec{x}, P(\vec{x}))$.

Как да построим програмата $P$?

# Семантика на Дийкстра

Разглеждаме програмите като **трансформатори на предикати** (predicate transformers).

**Дадено:** програма $P$, следусловие $R$.
**Търси се:** възможно най-слабото предусловие $Q := \mathtt{wp}(P, R)$, така че $\{Q\}P\{R\}$ да е валидна тройка на Хоар.

# Семантика на най-слабото предусловие

$$\mathtt{wp}(;, R) := R \qquad \text{(празен оператор)}$$

$$\mathtt{wp}(x = E, R) := R[x \mapsto E] \qquad \text{(присвояване)}$$

$$\mathtt{wp}(S; T, R) := \mathtt{wp}(S, \mathtt{wp}(T, R)) \qquad \text{(композиция)}$$

# Семантика на най-слабото предусловие

$$\texttt{wp}(;, R) := R \qquad \text{(празен оператор)}$$

$$\texttt{wp}(x = E, R) := R[x \mapsto E] \qquad \text{(присвояване)}$$

$$\texttt{wp}(S; T, R) := \texttt{wp}(S, \texttt{wp}(T, R)) \qquad \text{(композиция)}$$

# Семантика на най-слабото предусловие

$$\texttt{wp}(;, R) := R \qquad \text{(празен оператор)}$$

$$\texttt{wp}(x = E, R) := R[x \mapsto E] \qquad \text{(присвояване)}$$

$$\texttt{wp}(S; T, R) := \texttt{wp}(S, \texttt{wp}(T, R)) \qquad \text{(композиция)}$$

# Семантика на най-слабото предусловие

$$\texttt{wp}(\textbf{if}\ (B)\ S\ \textbf{else}\ T, R) := (B \to \texttt{wp}(S, R)) \wedge (\neg B \to \texttt{wp}(T, R))$$

(условен оператор)

$$\texttt{wlp}(\textbf{while}\ (B)\ S, R) := I \wedge (B \to \texttt{wlp}(S, I)) \wedge (\neg B \to R)$$

(итерация)

# Семантика на най-слабото предусловие

$$\texttt{wp}(\textbf{if}\ (B)\ S\ \textbf{else}\ T, R) := (B \to \texttt{wp}(S, R)) \wedge (\neg B \to \texttt{wp}(T, R))$$

(условен оператор)

$$\texttt{wlp}(\textbf{while}\ (B)\ S, R) := I \wedge (B \to \texttt{wlp}(S, I)) \wedge (\neg B \to R)$$

(итерация)

# И какво от това?

Какво като имаме такава семантика?

1. Свежда логиката на Хоар до предикатна логика.
2. Всъщност, системите за проверка на коректност работят с тази семантика, а не директно с логиката на Хоар!
3. Позволяват писане на програми, които са коректни по построение!

# И какво от това?

Какво като имаме такава семантика?

1. Свежда логиката на Хоар до предикатна логика.
2. Всъщност, системите за проверка на коректност работят с тази семантика, а не директно с логиката на Хоар!
3. Позволяват писане на програми, които са коректни по построение!

# И какво от това?

Какво като имаме такава семантика?

1. Свежда логиката на Хоар до предикатна логика.
2. Всъщност, системите за проверка на коректност работят с тази семантика, а не директно с логиката на Хоар!
3. Позволяват писане на програми, които са коректни по построение!

# И какво от това?

Какво като имаме такава семантика?

1. Свежда логиката на Хоар до предикатна логика.
2. Всъщност, системите за проверка на коректност работят с тази семантика, а не директно с логиката на Хоар!
3. Позволяват писане на програми, които са коректни по построение!

# Синтезиране на коректни програми

- Първо написваме спецификация на програмата
- Спецификацията представлява двойка от предусловие и следусловие $(Q, R)$
- Пишем програмата "отзад напред": избираме си оператор $S$ и пресмятаме $R' := \mathtt{wp}(S, R)$
- Всеки път като пишем цикъл, освен условие за коректност трябва да добавим и условие, че цикълът завършва
- Когато стигнем до ситуация, в която $Q \to R$, значи програмата е завършена и коректна :)
- Има системи, които позволяват полуавтоматично синтезиране на програми (B-Toolkit, Atelier B).

# Синтезиране на коректни програми

- Първо написваме спецификация на програмата
- Спецификацията представлява двойка от предусловие и следусловие $(Q, R)$
- Пишем програмата "отзад напред": избираме си оператор $S$ и пресмятаме $R' := \mathtt{wp}(S, R)$
- Всеки път като пишем цикъл, освен условие за коректност трябва да добавим и условие, че цикълът завършва
- Когато стигнем до ситуация, в която $Q \to R$, значи програмата е завършена и коректна :)
- Има системи, които позволяват полуавтоматично синтезиране на програми (B-Toolkit, Atelier B).

# Синтезиране на коректни програми

- Първо написваме спецификация на програмата
- Спецификацията представлява двойка от предусловие и следусловие $(Q, R)$
- Пишем програмата "отзад напред": избираме си оператор $S$ и пресмятаме $R' := \mathtt{wp}(S, R)$
- Всеки път като пишем цикъл, освен условие за коректност трябва да добавим и условие, че цикълът завършва
- Когато стигнем до ситуация, в която $Q \to R$, значи програмата е завършена и коректна :)
- Има системи, които позволяват полуавтоматично синтезиране на програми (B-Toolkit, Atelier B).

# Синтезиране на коректни програми

- Първо написваме спецификация на програмата
- Спецификацията представлява двойка от предусловие и следусловие $(Q, R)$
- Пишем програмата "отзад напред": избираме си оператор $S$ и пресмятаме $R' := \mathtt{wp}(S, R)$
- Всеки път като пишем цикъл, освен условие за коректност трябва да добавим и условие, че цикълът завършва
- Когато стигнем до ситуация, в която $Q \to R$, значи програмата е завършена и коректна :)
- Има системи, които позволяват полуавтоматично синтезиране на програми (B-Toolkit, Atelier B).

# Синтезиране на коректни програми

- Първо написваме спецификация на програмата
- Спецификацията представлява двойка от предусловие и следусловие $(Q, R)$
- Пишем програмата "отзад напред": избираме си оператор $S$ и пресмятаме $R' := \mathtt{wp}(S, R)$
- Всеки път като пишем цикъл, освен условие за коректност трябва да добавим и условие, че цикълът завършва
- Когато стигнем до ситуация, в която $Q \to R$, значи програмата е завършена и коректна :)
- Има системи, които позволяват полуавтоматично синтезиране на програми (B-Toolkit, Atelier B).

# Синтезиране на коректни програми

- Първо написваме спецификация на програмата
- Спецификацията представлява двойка от предусловие и следусловие $(Q, R)$
- Пишем програмата "отзад напред": избираме си оператор $S$ и пресмятаме $R' := \mathtt{wp}(S, R)$
- Всеки път като пишем цикъл, освен условие за коректност трябва да добавим и условие, че цикълът завършва
- Когато стигнем до ситуация, в която $Q \to R$, значи програмата е завършена и коректна :)
- Има системи, които позволяват полуавтоматично синтезиране на програми (B-Toolkit, Atelier B).

# Леко отклонение: какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

# Леко отклонение: какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

# Леко отклонение: какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

# Леко отклонение: какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

# Леко отклонение: какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

# Леко отклонение: какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

Стига глупости! Доказателството се пише на език с формален синтаксис, така че да може да се провери от машина!

# А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг
- да продаваме програмата, заедно с компютър, който я изпълнява

# А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг
- да продаваме програмата, заедно с компютър, който я изпълнява

# А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг
- да продаваме програмата, заедно с компютър, който я изпълнява

# А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг
- да продаваме програмата, заедно с компютър, който я изпълнява

# А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг
- да продаваме програмата, заедно с компютър, който я изпълнява

# А какво точно означава “да програмираме”?

Може да означава:

- да напишем пълния програмен код на лист
- да опишем идеята на алгоритъма в текстов файл
- да разпространяваме програмата като книжка с инструкции, разбираеми за човек
- да разпространяваме програмата като огромна разпечатка на машина на Тюринг
- да продаваме програмата, заедно с компютър, който я изпълнява

Стига глупости! Програмата се пише на програмен език с формален синтаксис, който може да се изпълни от машина!

# Леко отклонение: какво точно означава “да докажем”?

Може да означава:

- да напишем пълното математическо доказателство на лист
- да опишем идеята за доказателството в текстов файл
- да разпространяваме анотирана програма, която всеки да може да провери
- да разпространяваме анотирана програма заедно с огромно дърво построено по правилата на логиката на Хоар
- да разпространяваме програмата заедно със система за автоматична верификация

Стига глупости! Доказателството се пише на език с формален синтаксис, така че да може да се провери от машина!

# Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор
- автоматично доказване на теореми $\rightsquigarrow$ автоматично генериране на програми
- полуавтоматично доказване на теореми $\rightsquigarrow$ писане на програми с autocomplete

# Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор
- автоматично доказване на теореми $\rightsquigarrow$ автоматично генериране на програми
- полуавтоматично доказване на теореми $\rightsquigarrow$ писане на програми с autocomplete

# Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор
- автоматично доказване на теореми $\rightsquigarrow$ автоматично генериране на програми
- полуавтоматично доказване на теореми $\rightsquigarrow$ писане на програми с autocomplete

# Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор
- автоматично доказване на теореми $\rightsquigarrow$ автоматично генериране на програми
- полуавтоматично доказване на теореми $\rightsquigarrow$ писане на програми с autocomplete

# Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор
- автоматичо доказване на теореми $\rightsquigarrow$ автоматично генериране на програми
- полуавтоматично доказване на теореми $\rightsquigarrow$ писане на програми с autocomplete

# Формални доказателства

Теория на доказателствата е клон на математическата логика, който разглежда доказателствата като синтактични обекти.

- логическа система $\rightsquigarrow$ език за програмиране
- доказателство $\rightsquigarrow$ код на програма
- проверка за коректност $\rightsquigarrow$ компилация
- програма, проверяваща коректност $\rightsquigarrow$ компилатор
- автоматичо доказване на теореми $\rightsquigarrow$ автоматично генериране на програми
- полуавтоматично доказване на теореми $\rightsquigarrow$ писане на програми с autocomplete

# Примери за формални доказателства

http://wwwbroy.in.tum.de/~wenzelm/papers/romantic.pdf

# Системи за интерактивно доказване на теореми

Системите за доказване на теореми се делят на две категории:

- Системи за автоматично доказване на теореми (automated theorem provers)
- Системи за интерактивно доказване на теореми (interactive theorem provers, proof assistants

Изходът на система за доказване на теореми може да бъде:

- само информация за верността "вярно", "невярно", "не мога да определя" или просто да забива
- формално доказателство, ако теоремата е вярна
- контрапример, ако теоремата не е вярна

Формалното доказателство служи като сертификат за верността на дадена формула.

# Системи за интерактивно доказване на теореми

Системите за доказване на теореми се делят на две категории:

- Системи за автоматично доказване на теореми (automated theorem provers)
- Системи за интерактивно доказване на теореми (interactive theorem provers, proof assistants

Изходът на система за доказване на теореми може да бъде:

- само информация за верността “вярно”, “невярно”, “не мога да определя” или просто да забива
- формално доказателство, ако теоремата е вярна
- контрапример, ако теоремата не е вярна

Формалното доказателство служи като сертификат за верността на дадена формула.

# Системи за интерактивно доказване на теореми

Системите за доказване на теореми се делят на две категории:

- Системи за автоматично доказване на теореми (automated theorem provers)
- Системи за интерактивно доказване на теореми (interactive theorem provers, proof assistants

Изходът на система за доказване на теореми може да бъде:

- само информация за верността “вярно”, “невярно”, “не мога да определя” или просто да забива
- формално доказателство, ако теоремата е вярна
- контрапример, ако теоремата не е вярна

Формалното доказателство служи като сертификат за верността на дадена формула.

# Примери за системи за интерактивно доказване

ACL2, Agda, Coq, HOL Light, HOL4, Isabelle, LEGO, Matita, MINLOG, Mizar, NuPRL, PhoX, PVS, TPS, Twelf

# Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер
- Електронни ключалки
- Мрежови устройства
- Протоколи за сигурна комуникация

# Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер
- Електронни ключалки
- Мрежови устройства
- Протоколи за сигурна комуникация

# Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер
- Електронни ключалки
- Мрежови устройства
- Протоколи за сигурна комуникация

# Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер
- Електронни ключалки
- Мрежови устройства
- Протоколи за сигурна комуникация

# Сигурност на критични системи

- Софтуер и хардуер за космически апарати
- Медицински софтуер и хардуер
- Електронни ключалки
- Мрежови устройства
- Протоколи за сигурна комуникация

# Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

# Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

# Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

# Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

# Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

# Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

# Анализ на програми

- Автоматично генериране на unit тестове
- Намиране на бъгове чрез генериране на контрапримери
- Символно изпълнение и дебъгване
- Намиране на изтичане на информация
- Автоматично генериране на exploits
- Автоматично генериране на спецификация

Пример: Timsort

# Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на С програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за С програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за С
  - CompCert, проверен чрез Coq.

# Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на С програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за С програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за С
  - CompCert, проверен чрез Coq.

# Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на С програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за С програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за С
  - CompCert, проверен чрез Coq.

# Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на C програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за C програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за C
  - CompCert, проверен чрез Coq.

# Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на C програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за C програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за C
  - CompCert, проверен чрез Coq.

# Верифицирани компилатори

- Компилатори, които са доказано коректни.
- Т.е. генерираният машинен код има доказано същата семантика като тази на С програмата.
- Приложените оптимизации не променят смисъла на програмата.
- Ако сме доказали някакво свойство за С програмата, то важи и за изпълнимата програма.
- Вече има такъв компилатор за С
  - CompCert, проверен чрез Coq.

# Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.

Например:

- липса на memory leaks
- липса на buffer overflow
- липса на нежелано изтичане на информация
- коректност относно дадена спецификация

# Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.
Например:

- липса на memory leaks
- липса на buffer overflow
- липса на нежелано изтичане на информация
- коректност относно дадена спецификация

# Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.
Например:

- липса на memory leaks
- липса на buffer overflow
- липса на нежелано изтичане на информация
- коректност относно дадена спецификация

# Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.
Например:

- липса на memory leaks
- липса на buffer overflow
- липса на нежелано изтичане на информация
- коректност относно дадена спецификация

# Верифициращи компилатори

Компилатори, които могат да доказват хубави свойства на програмите, които компилират.
Например:

- липса на memory leaks
- липса на buffer overflow
- липса на нежелано изтичане на информация
- коректност относно дадена спецификация

# Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - "няма buffer overflow"
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

# Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - "няма buffer overflow"
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

# Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - "няма buffer overflow"
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

# Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - "няма buffer overflow"
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

# Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - “няма buffer overflow”
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

# Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - “няма buffer overflow”
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

# Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - “няма buffer overflow”
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

# Код, носещ доказателство

- Нека имаме дадена програма
- Написваме спецификация за нея
- Спецификацията може да не е пълна
  - например може да е някаква политика за сигурност
  - “няма buffer overflow”
- С помощта на система за доказване на теореми (автоматична или интерактивна) генерираме формално доказателство за коректност
- Пакетираме програмата, спецификацията и доказателството заедно
- Имаме ядро, което бързо и ефективно да провери доказателството и да пусне програмата само ако проверката е успешна.

# Код, носещ доказателство (2)

Ако злонамерен хакер се опита да промени кода, носещ доказателство:

- доказателството ще стане невалидно, тогава то ще бъде отхвърлено; или
- доказателството ще остане валидно, тогава програмата ще продължи да удовлетворява спецификацията си

# Сертифициращи компилатори

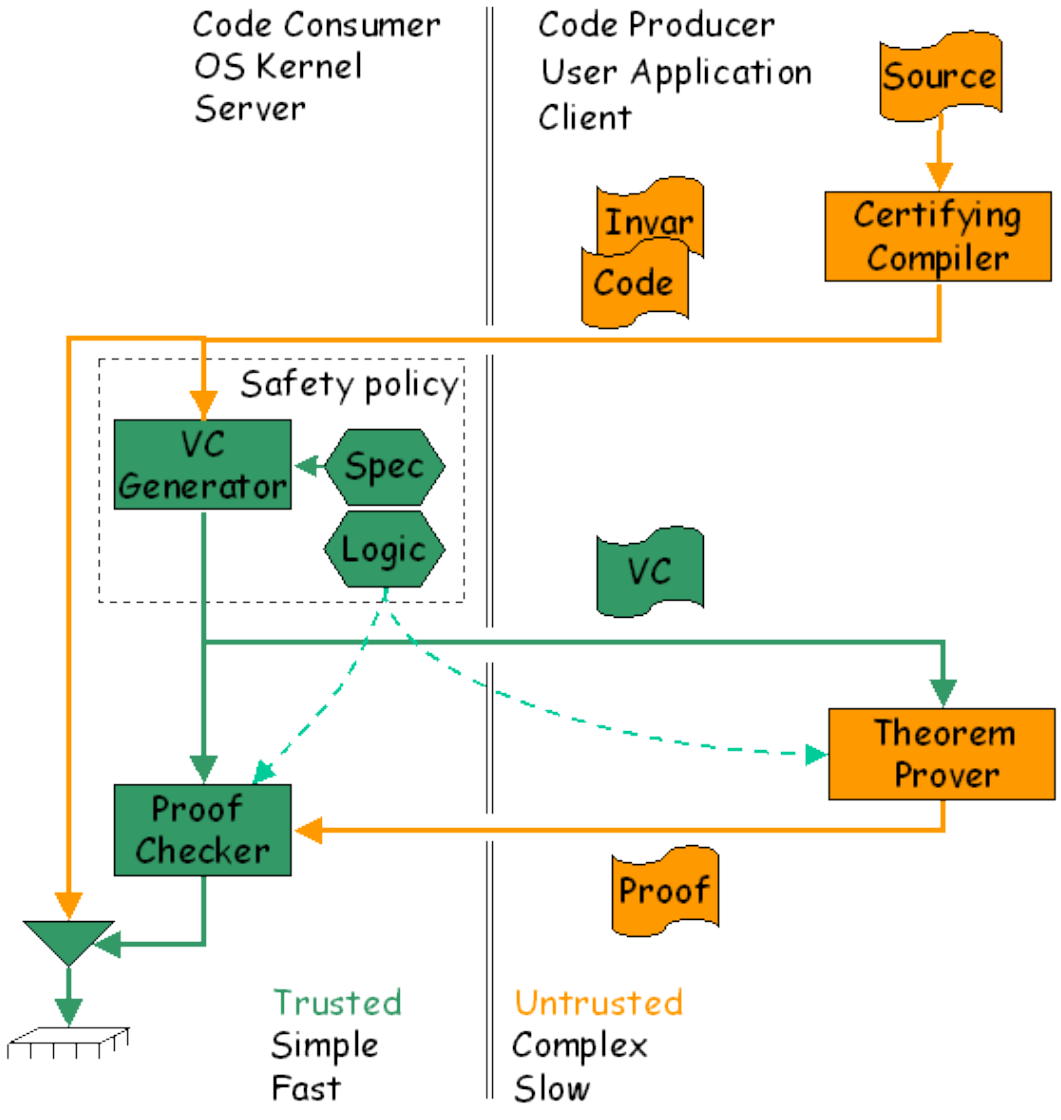
Сертифицираща компилатори са верифициращи компилатори, които генерират код, носещ доказателство.

Вече има такъв: Touchstone, сертифициращ компилатор за Java

# Сертифициращи компилатори

Сертифицираща компилатори са верифициращи компилатори, които генерират код, носещ доказателство.

Вече има такъв: Touchstone, сертифициращ компилатор за Java


Code Consumer
OS Kernel
Server
Code Producer
User Application
Client
Source
Invar
Code
Certifying
Compiler
Safety policy
VC
Generator
Spec
Logic
VC
Theorem
Prover
Proof
Checker
Proof
Trusted
Simple
Fast
Untrusted
Complex
Slow

Въпроси?