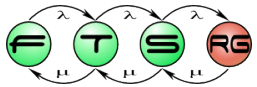
C programok formális verifikációja a Why3 keretrendszerrel

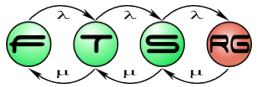
Izsó Benedek
benedek.izso@gmail.com

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék
Hibatűrő Rendszerek Kutatócsoport

Szoftver verifikáció és validáció



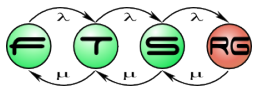
Tartalom



Why3 felépítése

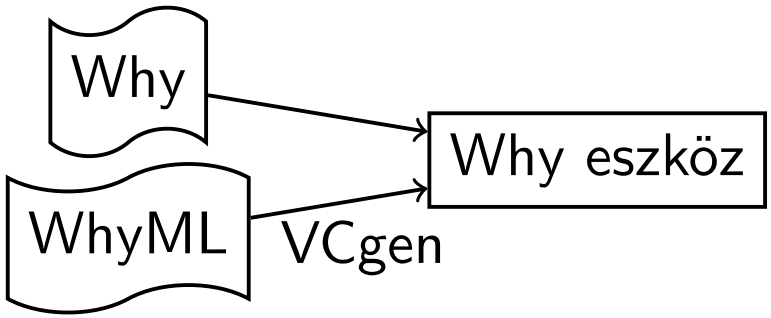
Why verifikációs eszköz

Why eszköz



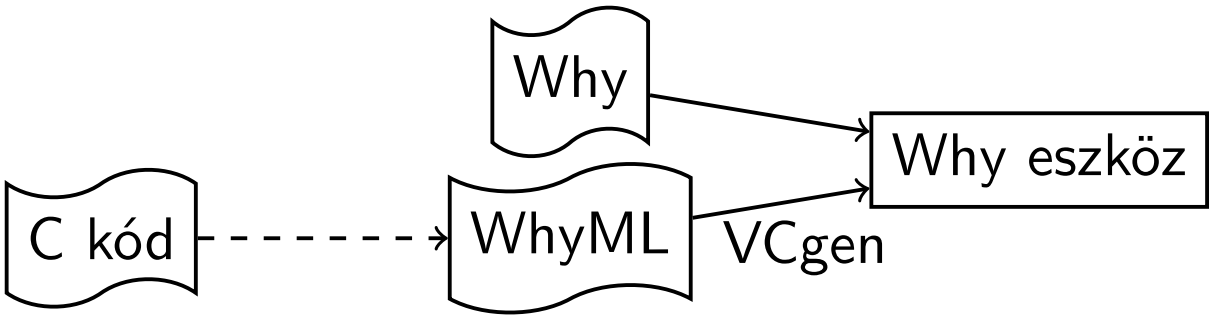
Why3 felépítése

Program leírás (**WhyML**) és logikai specifikáció (**Why**) elkülönítése



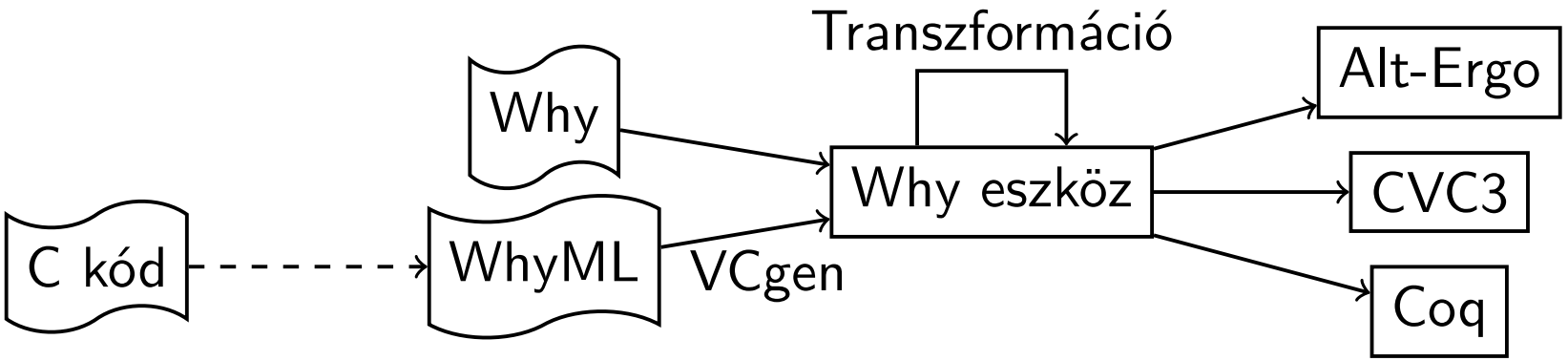
Why3 felépítése

C kód kézi formalizálása



Why3 felépítése

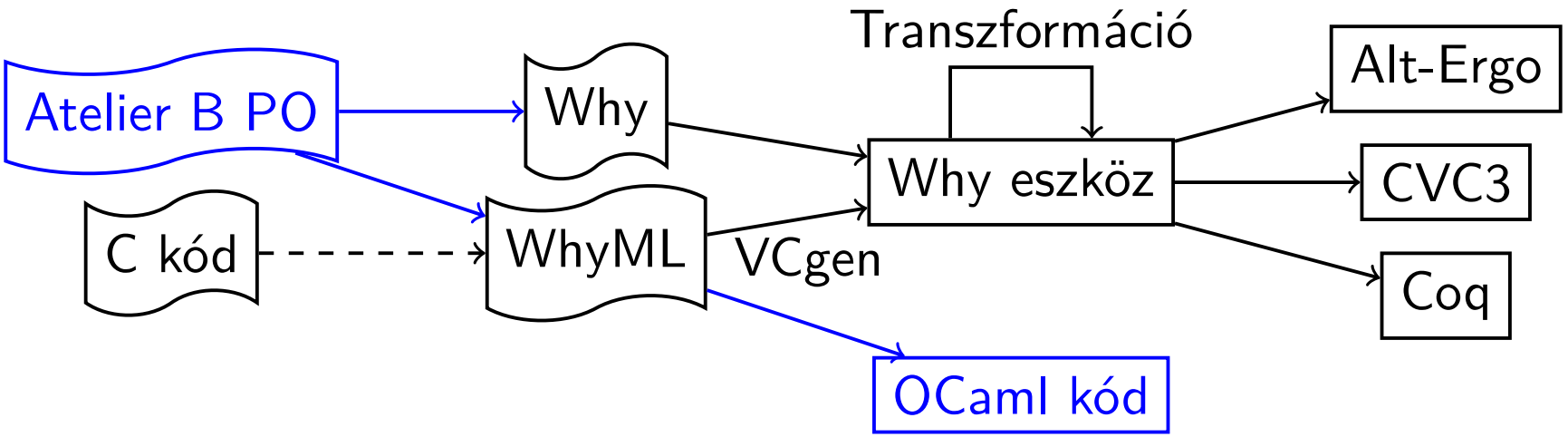
- Komplex kényszerek szétbontása transzformációval
- Több hatékony tétel bizonyító használata



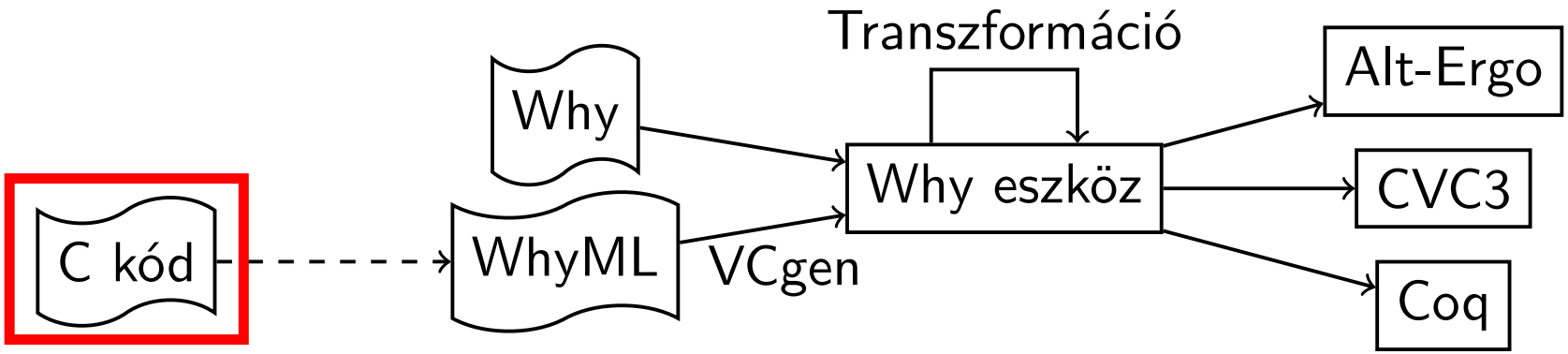
Why3 felépítése

Kiegészítés:

- Atelier B → Why
- WhyML → OCaml (correct by construction)



Forráskód megismerése



N-királynő probléma hatékony C kódja

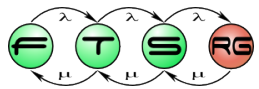
queens.c

```
t(a,b,c){int d=0,e=a&~b&~c,f=1;if(a)for(f=0;d=(e-=d)&-e;f+=t(a-d,(b+d)*2,(
c+d)/2));return f;}main(q){scanf("%d",&q);printf("%d\n",t(~(~0<<q),0,0));}
```

Definíció (N-királynő probléma)

Megadja, hogy N királynőt hány féle képpen lehet egy NxN-es sakktáblán elhelyezni.

```
$ gcc queens.c -o queens
$ ./queens
8
92
```



N-királynő probléma hatékony C kódja

Kód tördelése

```
t(a, b, c) {
  int d=0,e=a&~b&~c,f=1;
  if(a)
    for(f=0; d=(e-=d)&-e;
      f+=t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
main(q){
  scanf("%d",&q);
  printf("%d\n",t(~(~0<<q),0,0));
}
```

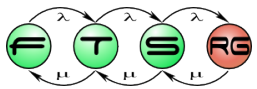
```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

N-királynő probléma hatékony C kódja

Típus információk hozzáadása

```
t(a, b, c) {
  int d=0,e=a&~b&~c,f=1;
  if(a)
    for(f=0; d=(e-=d)&-e;
      f+=t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
main(q){
  scanf("%d",&q);
  printf("%d\n",t(~(~0<<q),0,0));
}
```

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```



N-királynő probléma hatékony C kódja

ANSI C szerint mi az értéke?

```
t(a, b, c) {
  int d=0,e=a&~b&~c,f=1;
  if(a)
    for(f=0; d=(e-=d)&-e;
      f+=t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
main(q){
  scanf("%d",&q);
  printf("%d\n",t(~(~0<<q),0,0));
}
```

?

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

N-királynő probléma hatékony C kódja

ANSI C konform kód előállítása

```
t(a, b, c) {
  int d=0,e=a&~b&~c,f=1;
  if(a)
    for(f=0; d=(e-=d)&-e;
      f+=t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
main(q){
  scanf("%d",&q);
  printf("%d\n",t(~(~0<<q),0,0));
}
```

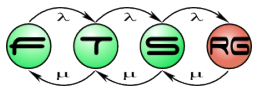
```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

N-királynő probléma hatékony C kódja

I/O nem érdekes, csak az algoritmus

```
t(a, b, c) {
  int d=0,e=a&~b&~c,f=1;
  if(a)
    for(f=0; d=(e-=d)&-e;
      f+=t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
main(q){
  scanf("%d",&q);
  printf("%d\n",t(~(~0<<q),0,0));
}
```

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,(c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

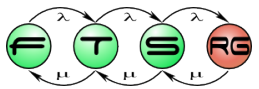


Algoritmus leírása halmazokkal

Visszalépéses keresés (backtracking):
- t() helyezi el a sorokban a királynőket
- a,b,c: {0,1,...,n-1} részhalmazát jelöli, *i* eleme a halmaznak, ha az *i*. bit 1

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,
             (c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

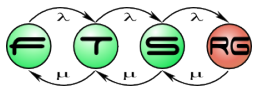


Algoritmus leírása halmazokkal

Bitművelet → halmazművelet:

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,
             (c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

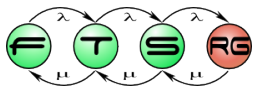


Algoritmus leírása halmazokkal

Bitművelet → halmazművelet:
- Halmaz kivonás

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,
             (c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

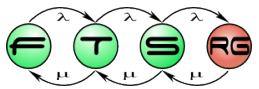


Algoritmus leírása halmazokkal

Bitművelet → halmazművelet:
- Halmaz kivonás
- Üreshalmaz vizsgálat

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,
             (c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```



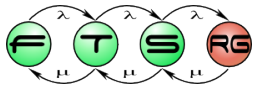
Algoritmus leírása halmazokkal

Bitművelet → halmazművelet:
- Halmaz kivonás
- Üreshalmaz vizsgálat
- Legkisebb elem kiválasztása

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,
             (c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

011010 = e
100110 = $-e$ (kettes kompl.)
000010 = $e\& - e$



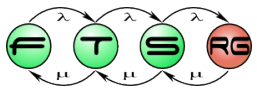
Algoritmus leírása halmazokkal

Bitművelet → halmazművelet:
- *a*-ban *szereplő d* elem törlése

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,
             (c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

$a \supset e \supset d$



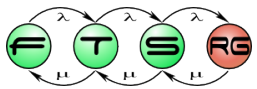
Algoritmus leírása halmazokkal

Bitművelet → halmazművelet:
- *a*-ban *szereplő d* elem törlése
- *b* és *c*-ben nem szereplő új elem felvétele

```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,
             (c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

$d \notin b$
$d \notin c$

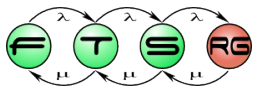


Algoritmus leírása halmazokkal

Bitművelet → halmazművelet:
- *a*-ban *szereplő d* elem törlése
- *b* és *c*-ben nem szereplő új elem felvétele
- Minden elem növelése/csökkentése

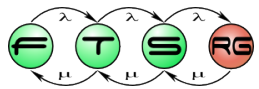
```
int t(int a, int b, int c) {
  int d, e=a&~b&~c, f=1;
  if (a)
    for (f=0; d=e&-e; e-=d)
      f += t(a-d,(b+d)*2,
             (c+d)/2));
  return f;
}
int queens(int n) {
  return t(~(~0<<n),0,0);
}
```

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```



Algoritmus megértése

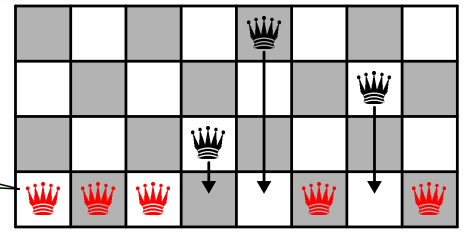
```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f

int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```



Algoritmus megértése

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f

int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

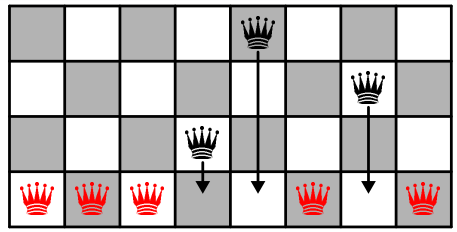
a:
függőlegesen nem fogott lehetséges elemek



$\mathtt{a} = 11100101_2$, $a = \{0, 2, 5, 6, 7\}$

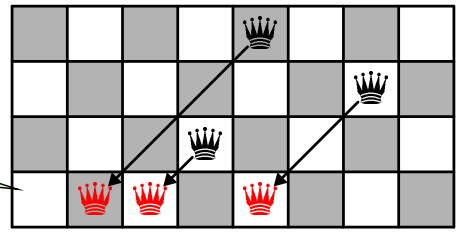
Algoritmus megértése

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f

int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```



a $= 11100101_2$, $a = \{0, 2, 5, 6, 7\}$

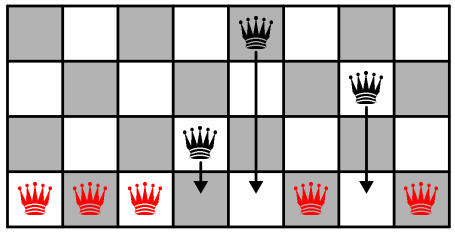
b:
balra átlósan
fogott elemek



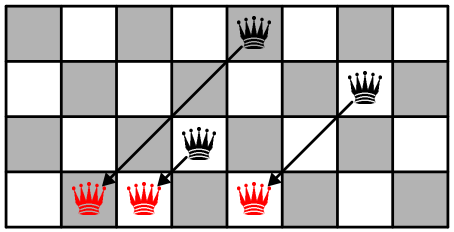
b $= 01101000_2$, $b = \{3, 5, 6\}$

Algoritmus megértése

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f

int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

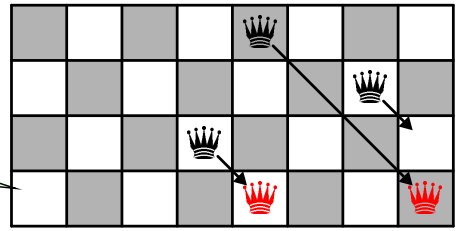


a $= 11100101_2$, $a = \{0, 2, 5, 6, 7\}$



b $= 01101000_2$, $b = \{3, 5, 6\}$

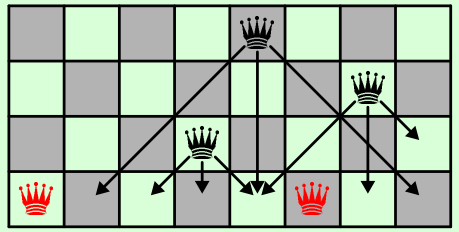
c:
jobbra átlósan
fogott elemek



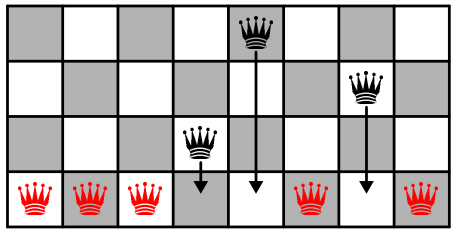
c $= 00001001_2$, $c = \{0, 3\}$

Algoritmus megértése

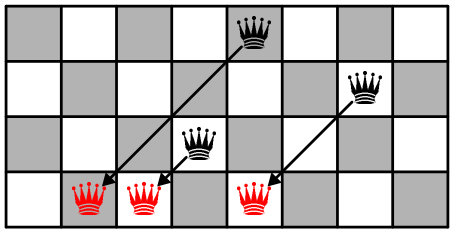
```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f

int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```



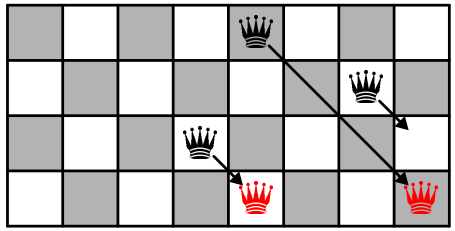
a&~b&~c = 10000100_2, $a \setminus b \setminus c = \{2, 7\}$



a = 11100101_2, $a = \{0, 2, 5, 6, 7\}$



b = 01101000_2, $b = \{3, 5, 6\}$

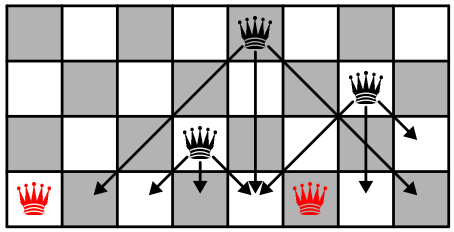


c = 00001001_2, $c = \{0, 3\}$

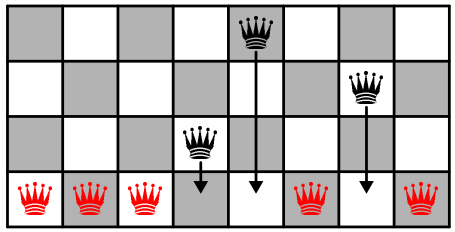
Algoritmus megértése

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f

int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

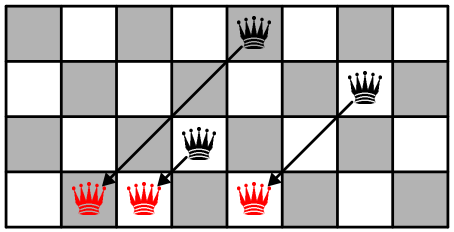
Ciklus:
i) e- legkisebb elemének kiválasztása d-be
ii) többi sor feltöltése rekurzívan
iii) elem kivétele lehetségesek közül



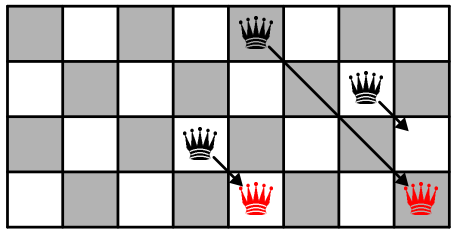
a&~b&~c $= 10000100_2$, $a \setminus b \setminus c = \{2, 7\}$



a $= 11100101_2$, $a = \{0, 2, 5, 6, 7\}$



b $= 01101000_2$, $b = \{3, 5, 6\}$



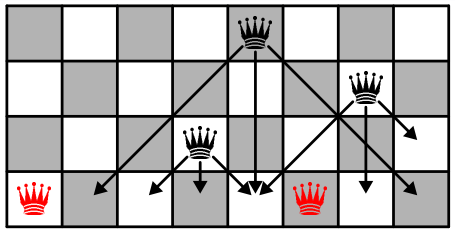
c $= 00001001_2$, $c = \{0, 3\}$

Algoritmus megértése

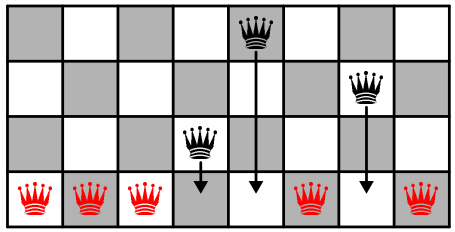
```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f

int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

$a = \emptyset$ akkor új megoldás

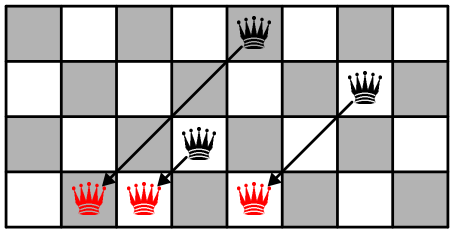
a minden lehetséges helyet tartalmaz
b és *c* üres



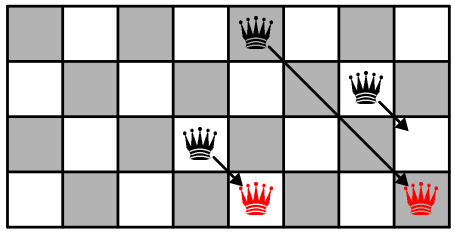
a&~b&~c $= 10000100_2$, $a \setminus b \setminus c = \{2, 7\}$



a $= 11100101_2$, $a = \{0, 2, 5, 6, 7\}$

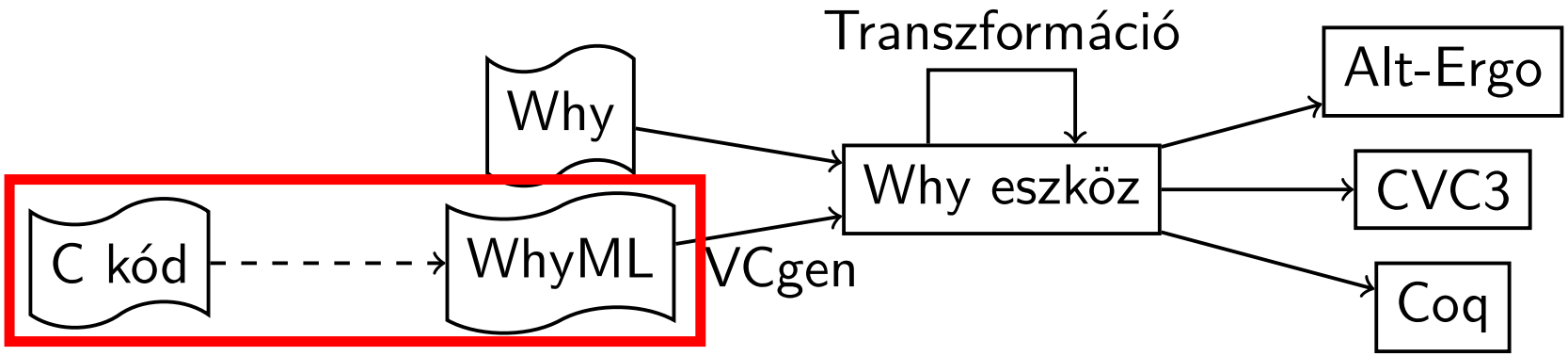


b $= 01101000_2$, $b = \{3, 5, 6\}$



c $= 00001001_2$, $c = \{0, 3\}$

Algoritmus WhyML nyelven



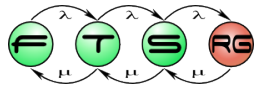
Algoritmus WhyML nyelven

```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
```

```
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      f := !f + t (remove d a) (succ (add d b))
                               (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1
```

```
let queens (q: int) =
  t (below q) empty empty
```



Algoritmus WhyML nyelven

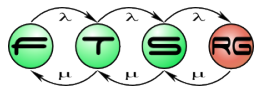
```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
```

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      f := !f + t (remove d a) (succ (add d b))
                               (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1
```

```
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

```
let queens (q: int) =
  t (below q) empty empty
```

WhyML nyelv:
- (rekurzív) függvény definiálása a **let (rec)** kulcsszóval



Algoritmus WhyML nyelven

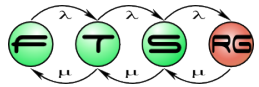
```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
```

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      f := !f + t (remove d a) (succ (add d b))
                               (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1
```

```
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

```
let queens (q: int) =
  t (below q) empty empty
```

WhyML nyelv:
- (rekurzív) függvény definiálása a **let (rec)** kulcsszóval
 - **ref**: értékre mutató referencia, **!f**: f értéke, **:=** értékadás



Algoritmus WhyML nyelven

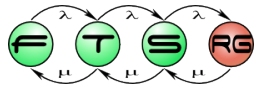
```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
```

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      f := !f + t (remove d a) (succ (add d b))
                               (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1
```

```
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

```
let queens (q: int) =
  t (below q) empty empty
```

WhyML nyelv:
- (rekurzív) függvény definiálása a **let (rec)** kulcsszóval
  - **ref**: értékre mutató referencia, **!f**: f értéke, **:=** értékadás
  - **halmaz** és **egész szám** műveletek beépített könyvtárból



Algoritmus WhyML nyelven

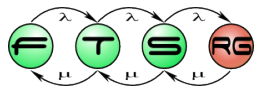
```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
```

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      f := !f + t (remove d a) (succ (add d b))
                               (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1
```

```
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

```
let queens (q: int) =
  t (below q) empty empty
```

WhyML nyelv:
- (rekurzív) függvény definiálása a **let (rec)** kulcsszóval
  - **ref**: értékre mutató referencia, **!f**: f értéke, **:=** értékadás
  - **halmaz** és **egész szám** műveletek beépített könyvtárból
  - vezérlési szerkezetek (**if-else, while**)



Algoritmus WhyML nyelven

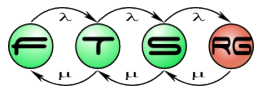
```
int t(set a, set b, set c)
  f ← 1
  if a ≠ ∅
    e ← (a \ b) \ c
    f ← 0
    while e ≠ ∅
      d ← min_elt(e)
      f ← f + t(a\{d}, succ(b∪{d}),
                pred(c∪{d}))
      e ← e \ {d}
  return f
```

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      f := !f + t (remove d a) (succ (add d b))
                               (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1
```

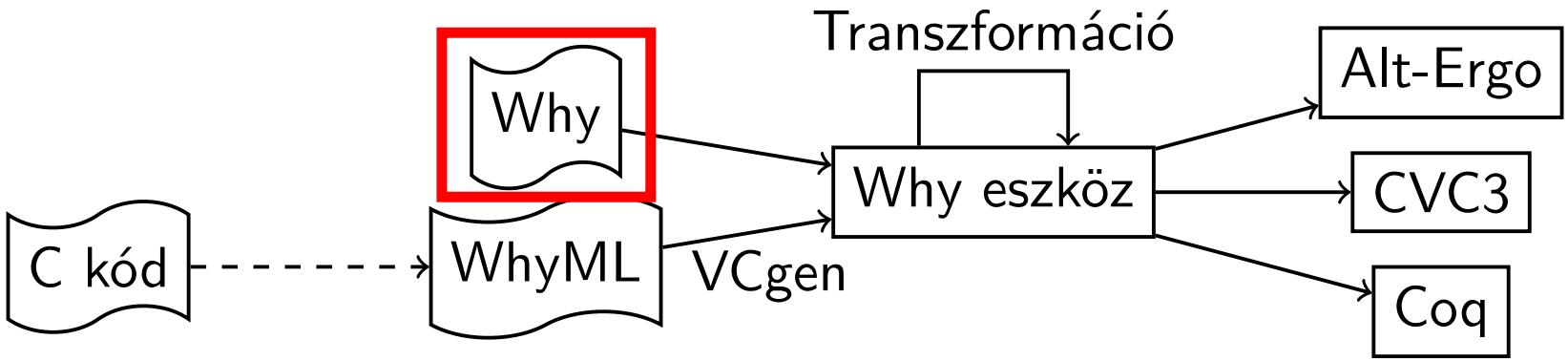
```
int queens(int n)
  return t({0, 1, ..., n−1}, ∅, ∅)
```

```
let queens (q: int) =
  t (below q) empty empty
```

```
axiom succ_def:
      ∀ s: set int, i: int, mem i (succ s) ↔ i ≥ 1 ∧ mem (i−1) s
axiom succ_def:
      ∀ s: set int, i: int, mem i (succ s) ↔ i ≥ 1 ∧ mem (i−1) s
```



Ellenőrzendő kritériumok

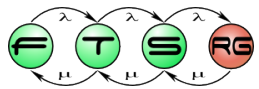


Ellenőrzendő kritériumok

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in


      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));

      e := remove d !e
    done;
    !f
  end else


    1
let queens (q: int) =

  t (below q) empty empty
```

Ellenőrzendő kritériumok:

1. Nem szakad meg a futása
2. Véges időn belül lefut
3. Helyes eredményt ad



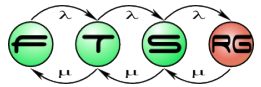
Ellenőrzendő kritériumok

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in


      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));

      e := remove d !e
    done;
    !f
  end else


    1
let queens (q: int) =

  t (below q) empty empty
```

Nincs kritikus utasítás
pl.: osztás, tömb címzés

Ellenőrzendő kritériumok:

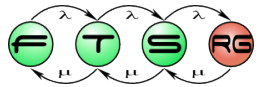
1. Nem szakad meg a futása ✓
2. Véges időn belül lefut
3. Helyes eredményt ad



Ellenőrzendő kritériumok

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in


      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));

      e := remove d !e
    done;
    !f
  end else


    1
let queens (q: int) =

  t (below q) empty empty
```

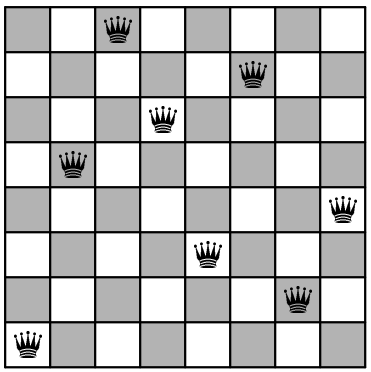
Ellenőrzendő kritériumok:
milyen adatokra?
→ új változók bevezetése
- függhet a program változóitól
- program változó nem függet tőle
- hatékonyság nem számít



Ellenőrzendő kritériumok

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      (* ghost *) col := !col[!k ← d];
      (* ghost *) incr k;
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      (* ghost *) decr k;
      e := remove d !e
    done;
    !f
  end else
    (* ghost *) sol := !sol[!s ← !col];
    (* ghost *) incr s;
    1
let queens (q: int) =

  t (below q) empty empty
```

```
type solution = map int int
val col: ref solution
val k: ref int

type solutions = map int solutions
val sol: ref solutions
val s: ref int
```

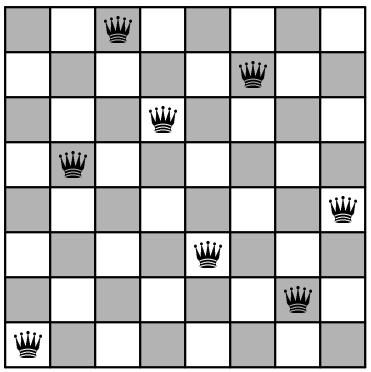


col={5,2,4,6,0,3,1,7}, k=8

Ellenőrzendő kritériumok

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      (* ghost *) col := !col[!k ← d];
      (* ghost *) incr k;
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      (* ghost *) decr k;
      e := remove d !e
    done;
    !f
  end else
    (* ghost *) sol := !sol[!s ← !col];
    (* ghost *) incr s;
    1
let queens (q: int) =
  { 0 ≤ q = n ∧ !s = 0 ∧ !k = 0 }
  t (below q) empty empty
  { result = !s ∧ sorted !sol 0 !s ∧
    ∀ u: solution. solution u ↔
        (∃ i:int. 0 ≤ i < result ∧ eq_sol u !sol[i]) }
```

```
type solution = map int int
val col: ref solution
val k: ref int

type solutions = map int solutions
val sol: ref solutions
val s: ref int
```

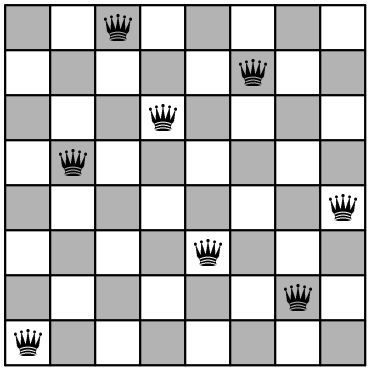


col={5,2,4,6,0,3,1,7}, k=8

Ellenőrzendő kritériumok

```
let rec t (a b c: set int) =
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      let d = min_elt !e in
      (* ghost *) col := !col[!k ← d];
      (* ghost *) incr k;
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      (* ghost *) decr k;
      e := remove d !e
    done;
    !f
  end else
    (* ghost *) sol := !sol[!s ← !col];
    (* ghost *) incr s;
    0
```

```
type solution = map int int
val col: ref solution
val k: ref int

type solutions = map int solutions
val sol: ref solutions
val s: ref int
```



col={5,2,4,6,0,3,1,7}, k=8

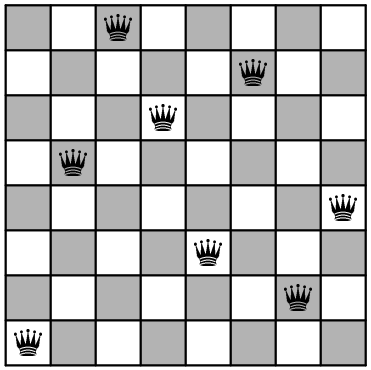
Az eredmény a megoldások száma

Egy eredmény egyszer szerepel

Csakis az megoldás ami sol-ban megtalálható

```
t (below q) empty empty
{ result = !s ∧ sorted !sol 0 !s ∧
  ∀ u: solution. solution u ↔
      (∃ i:int. 0 ≤ i < result ∧ eq_sol u !sol[i]) }
```

Ellenőrzendő kritériumok



col={5,2,4,6,0,3,1,7}, k=8

```
let queens (q: int) =
  { 0 ≤ q = n ∧ !s = 0 ∧ !k = 0 }
  t (below q) empty empty
  { result = !s ∧ sorted !sol 0 !s ∧
    ∀ u: solution. solution u ↔
        (∃ i:int. 0 ≤ i < result ∧ eq_sol u !sol[i]) }
```

function n : int

n paraméter függvényként

predicate partial_solution (k: int) (s: solution) =
  ∀ i: int. 0 ≤ i < k →
  0 ≤ s[i] < n ∧
  (∀ j: int. 0 ≤ j < i →
    s[i] ≠ s[j] ∧ s[i]-s[j] ≠ i-j ∧ s[i]-s[j] ≠ j-i)

- s k hosszon részleges megoldás, ha
- minden i. helyen táblán lévő index szerepel, és
- az előző (j) helyek semelyike sincs:
  egy oszlopban, jobbra átlósan, balra átlósan

predicate solution (s: solution) =
  partial_solution n s

s megoldás,
ha n hosszon részleges megoldás

Ellenőrzendő kritériumok

0	4	7	5	2	6	1	3
0	5	7	2	6	3	1	4
0	6	3	5	7	1	4	2
0	6	4	7	1	3	5	2
1	3	5	7	2	0	6	4
			⋮				

sol elemei

```
let queens (q: int) =
  { 0 ≤ q = n ∧ !s = 0 ∧ !k = 0 }
  t (below q) empty empty
  { result = !s ∧ sorted !sol 0 !s ∧
    ∀ u: solution. solution u ↔
        (∃ i:int. 0 ≤ i < result ∧ eq_sol u !sol[i]) }
```

predicate eq_prefix (t u: map int α) (i: int) =
  ∀ k: int. 0 ≤ k < i → t[k] = u[k]

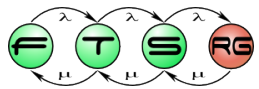
t és u i hosszan azonos prefixű, ha ott (k<i) értékeik megegyeznek

predicate lt_sol (s1 s2: solution) =
  ∃ i: int. 0 ≤ i < n ∧
  eq_prefix s1 s2 i ∧ s1[i] < s2[i]

s1 s2 megoldás előtt szerepel, ha egy indexen kisebb s1 értéke és előtte azonos volt a prefixük

predicate sorted (s: solutions) (a b: int) =
  ∀ i j: int. a ≤ i < j < b → lt_sol s[i] s[j]

megoldások rendezettek, ha minden j-nél kisebb i indexre i. megoldás a j. előtt van



Ellenőrzendő kritériumok

0	4	7	5	2	6	1	3
0	5	7	2	6	3	1	4
0	6	3	5	7	1	4	2
0	6	4	7	1	3	5	2
1	3	5	7	2	0	6	4
			⋮				

sol elemei

```
let queens (q: int) =
  { 0 ≤ q = n ∧ !s = 0 ∧ !k = 0 }
  t (below q) empty empty
  { result = !s ∧ sorted !sol 0 !s ∧
    ∀ u: solution. solution u ↔
        (∃ i:int. 0 ≤ i < result ∧ eq_sol u !sol[i]) }
```

predicate eq_prefix (t u: map int α) (i: int) =
  ∀ k: int. 0 ≤ k < i → t[k] = u[k]

polimorf

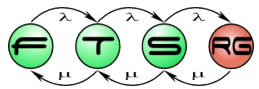
t és u i hosszan azonos prefixű, ha ott (k<i) értékeik megegyeznek

predicate lt_sol (s1 s2: solution) =
  ∃ i: int. 0 ≤ i < n ∧
  eq_prefix s1 s2 i ∧ s1[i] < s2[i]

s1 s2 megoldás előtt szerepel, ha egy indexen kisebb s1 értéke és előtte azonos volt a prefixük

predicate sorted (s: solutions) (a b: int) =
  ∀ i j: int. a ≤ i < j < b → lt_sol s[i] s[j]

megoldások rendezettek, ha minden j-nél kisebb i indexre i. megoldás a j. előtt van



Ellenőrzendő kritériumok

0	4	7	5	2	6	1	3
0	5	7	2	6	3	1	4
0	6	3	5	7	1	4	2
0	6	4	7	1	3	5	2
1	3	5	7	2	0	6	4
			⋮				

sol elemei

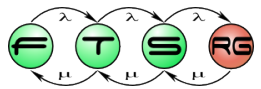
```
let queens (q: int) =
  { 0 ≤ q = n ∧ !s = 0 ∧ !k = 0 }
  t (below q) empty empty
  { result = !s ∧ sorted !sol 0 !s ∧
    ∀ u: solution. solution u ↔
        (∃ i:int. 0 ≤ i < result ∧ eq_sol u !sol[i]) }
```

```
predicate eq_prefix (t u: map int α) (i: int) =
  ∀ k: int. 0 ≤ k < i → t[k] = u[k]
```

t és u i hosszan azonos prefixű, ha ott (k<i) értékeik megegyeznek

```
predicate eq_sol (t u: solution) =
  eq_prefix t u n
```

t és u megoldás azonos, ha n hosszon ugyanaz a prefixük

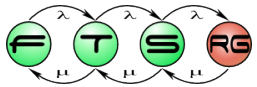


Megállás biztosítása

```
let rec t (a b c: set int)

=
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do


      let d = min_elt !e in
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      e := remove d !e
    done;
   !f
  end else
    1

let queens (q: int) =
  t (below q) empty empty
```

Megállás biztosítása:

1. A rekurzív t() megáll véges időn belül
2. A while ciklus megáll véges időn belül



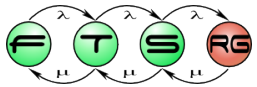
Megállás biztosítása

```
let rec t (a b c: set int)
variant { cardinal a }
=
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      variant { cardinal !e }
      invariant { subset !e a }
      let d = min_elt !e in
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1

let queens (q: int) =
  t (below q) empty empty
```

- variant: szig. mon. csökkenő kifejezés egy jól megalapozott rendezést tekintve
- invariant: folyamatosan igaz kifejezés

- cardinal: halmaz számossága
- subset: részhalmaza-e

A cikluson belül tudott, hogy d ⊆ e, de `remove d a`-hoz elő kell írni, hogy e ⊆ a



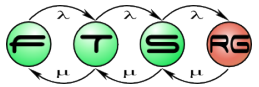
Megállás biztosítása

```
let rec t (a b c: set int)
variant { cardinal a }
=
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do
      variant { cardinal !e }
      invariant { subset !e (diff (diff a b) c) }
      let d = min_elt !e in
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1

let queens (q: int) =
  t (below q) empty empty
```

- variant: szig. mon. csökkenő kifejezés egy jól megalapozott rendezést tekintve
- invariant: folyamatosan igaz kifejezés

- cardinal: halmaz számossága
- subset: részhalmaza-e

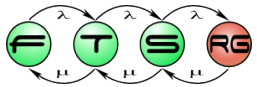
A cikluson belül tudott, hogy d $\subseteq$ e, de `remove d a`-hoz elő kell írni, hogy e $\subseteq$ a



t függvény specifikációja q helyességéhez

```
let rec t (a b c: set int)


=
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do

      let d = min_elt !e in
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      e := remove d !e
    done;
   !f
  end else
    1

let queens (q: int) =
  t (below q) empty empty
```

q specifikációja
t feltételei miatt teljesülnek
$\rightarrow$
szükség van t specifikációjára



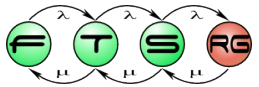
t függvény specifikációja q helyességéhez

```
let rec t (a b c: set int)
requires { 0 ≤ !k ∧ !k + cardinal a = n ∧ !s ≥ 0 ∧ ...}
ensures { result = !s - old !s ≥ 0 ∧ !k = old !k ∧ ...
          sorted !sol (old !s) !s ∧ ...}
=
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
    while not (is_empty !e) do

      let d = min_elt !e in
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1

let queens (q: int) =
  t (below q) empty empty
```

q specifikációja
t feltételei miatt teljesülnek
→
szükség van t specifikációjára

Requires (előfeltételek):
- eddig elhelyezett és nem elhelyezett elemek száma n

Ensures (utófeltételek):
- visszatérési értékkel nő a megoldások száma
- k nem változik
- sol-ba beszúrt megoldás rendezett



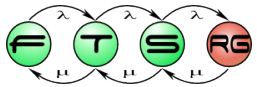
t függvény specifikációja q helyességéhez

```
let rec t (a b c: set int)
requires { 0 ≤ !k ∧ !k + cardinal a = n ∧ !s ≥ 0 ∧ ...}
ensures { result = !s - old !s ≥ 0 ∧ !k = old !k ∧ ...
          sorted !sol (old !s) !s ∧ ...}
=
  if not (is_empty a) then begin
    let e = ref (diff (diff a b) c) in
    let f = ref 0 in
  'L:while not (is_empty !e) do
      invariant{ !f = !s - at !s 'L ≥ 0 ∧ !k = at !k 'L }
      let d = min_elt !e in
      f := !f + t (remove d a)
                  (succ (add d b))
                  (pred (add d c));
      e := remove d !e
    done;
    !f
  end else
    1

let queens (q: int) =
  t (below q) empty empty
```

függvény utófeltételéhez ciklus invariáns felvétele

- 'L: címke
- at !s 'L: s értéke 'L helyen

A ciklus futása során:
- f értéke az új megoldások számát tartalmazza
- ami nem negatív
- k nem változik



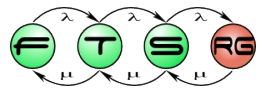
t függvény specifikációja q helyességéhez

```
let rec t (a b c: set int)
  requires { 0 ≤ !k ∧ !k + cardinal a = n ∧ !s ≥ 0 ∧ ...}
  ensures { result = !s − old !s ≥ 0 ∧ !k − old !k ∧
            sorted !sol (old !s) !s ∧ ...}
=
  if not (is_empty a) then begin
    let f = ref (min_elt a) in
    let d = ref (diff a b) in
    let e = ref (min_elt !d) in
    d := remove !e !d;
    !f
  end else
    1

let queens (q: int) =
  t (below q) empty e
```

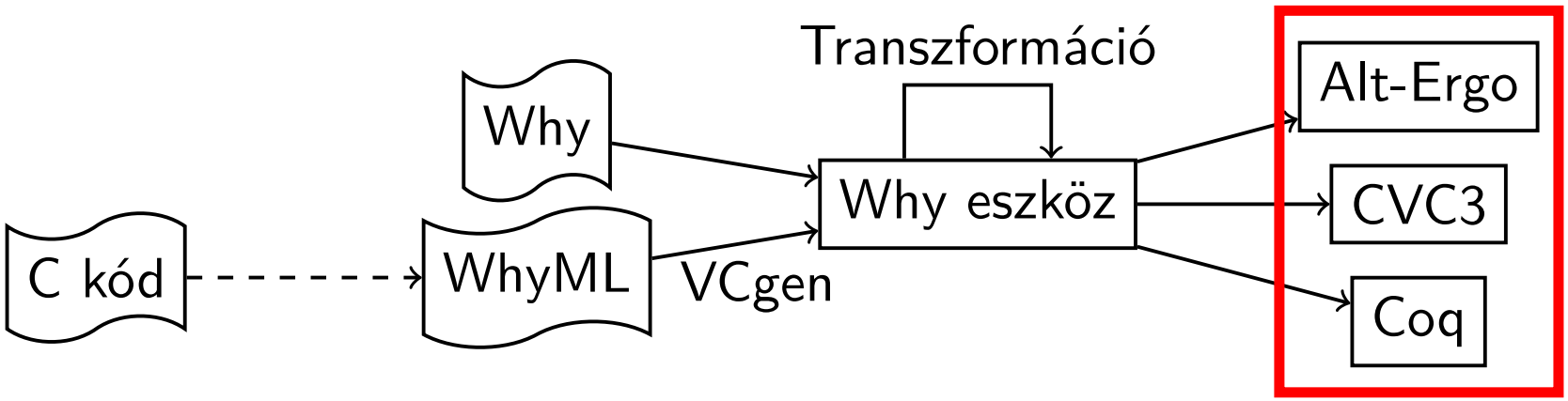
```
invariant {
  !f = !s − at !s 'L >= 0 /\ !k = at !k 'L /\
  subset !e (diff (diff a b) c) /\
  partial_solution !col /\
  sorted !sol (at !s 'L) /\
  (forall i j: int. 0 <= i < j < at !s 'L ->
   (solution t /\ eq_prefix !col t !k) <->
   (exists i: int. 0 <= i < !s /\ eq_sol t !sol[i])) /\
  (* assigns *)
  eq_prefix (at !col 'L) !col !k /\
  eq_prefix (at !sol 'L) !sol !s
```

```
ensures { result = !s − old !s >= 0 /\ !k = old !k /\
  sorted !sol (old !s) !s /\
  (forall t: solution.
   (solution t /\ eq_prefix !col t !k) <->
   (exists i: int. old !s <= i < !s /\ eq_sol t !sol[i])) /\
  (* assigns *)
  eq_prefix (old !col) !col !k /\
  eq_prefix (old !sol) !sol (old !s) }
```

```
requires { 0 <= !k <= n /\ !s >= 0 /\
  (forall i: int. 0 <= i < !s -> 0 <= !col[i] < n) /\
  (forall i j: int. 0 <= i < j < !s -> !col[i] <> !col[j] /\
   !col[i] - !col[j] <> j - i /\ !col[j] - !col[i] <> j - i) /\
  (forall i: int. 0 <= i < !s -> !col[i] <> !k /\
   !col[i] - !k <> !s - i /\ !k - !col[i] <> !s - i) /\
  (forall j: int. 0 <= j < !k ->
   (forall i: int. 0 <= i < !s -> !col[i] <> j /\
    !col[i] - j <> !s - i /\ j - !col[i] <> !s - i)) /\
  partial_solution !k !col /\
  sorted !sol (old !s) !s /\
  (forall t: solution.
   ((solution t /\ eq_prefix !col t !k) <->
    (exists i: int. old !s <= i < !s /\ eq_sol t !sol[i]))
```

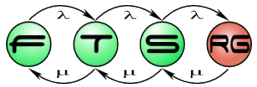


Bizonyítás futtatása



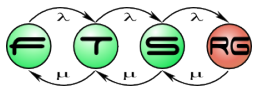
Bizonyítás futtatása

- 2 sor C kód $\rightarrow$ 46 sor kód; feltételekkel 160
  ügyes algoritmus, összetett feltételek
- 43 bizonyítandó tétel: 37 automatikusan bizonyított
  (Alt-Ergo, CVC3)
- 6 felhasználói segítséggel (Coq), ami néhány óra munkát jelent



Összefoglalás

- C program formális verifikációja (helyesség, terminálás)
- Algoritmus kézi megfogalmazása (WhyML nyelven), de hasonló programozási fogalmak, vezérlési struktúrák támogatottak
- Specifikáció megfogalmazása Why nyelven
- Automatikus és kézi tételbizonyítók kevert használata
- Túlcsordulás kezelése: tetszőlegesen nagy int típus (lassú) / n korlátozása (64 biten $n \leq 28$)



Irodalomjegyzék

Jean-Christophe Filliâtre
Verifying Two Lines of C with Why3: an Exercise in Program Verification.
VSTTE, 2012.

François Bobot and Jean-Christophe Filliâtre and Claude Marché and Andrei Paskevich
Why3: Shepherd Your Herd of Provers.
Boogie, 2011

Mentré, David and Marché, Claude and Filliâtre, Jean-Christophe and Asuka, Masashi
Discharging Proof Obligations from Atelier B using Multiple Automated Provers.
Springer, 2012