

Programhelyesség-bizonyítás

Majzik István

Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

<http://www.mit.bme.hu/~majzik/>

Motiváció

- Kritikus algoritmusok helyességének bizonyítása
 - Korlátozott feladatok: biztonságkritikus funkciók, hozzáférésvédelem, protokoll mag, ...
 - Bizonyítás alapja:
 - **Részletes terv**: Pszeudo-nyelven leírt algoritmus (a továbbiakban mint „program”)
 - **Valós programnyelv**: Csak részleges támogatás van
- Tételbizonyító rendszerek használata
 - Bizonyítandó tulajdonság mint bizonyítandó „tétel”
 - Klasszikus elő- és utófeltételek a tipikusak (contract)
- Kihívások:
 - (Pszeudo) programból hogyan lesz bizonyítandó tétel?
 - Milyen **stratégiák** használhatók a bizonyításhoz?

Tételbizonyító rendszerek

- Felépítés:

- Leíró nyelv, pl.:
 - Elsőrendű logika
 - Típusokkal kiegészített logikák
 - Magasabbrendű logika, ...
- Problématér leírása: logikai axiómák
- Megoldandó feladat: bizonyítandó tétel
- Következtetési szabályok (levezetési szabályok)
 - Indukció, dedukció, unifikáció, ...

- Komponensek:

- **Algoritmikus:** Egy-egy következtetési szabály alkalmazása
- **Kereső:** Stratégia/taktika a szabályok kiválasztására
 - Cél-vezérelt (visszafelé történő) keresés
 - Mélységi vagy szélességi keresés
 - Interaktív (felhasználói segítséggel)

Tételbizonyító rendszerek alkalmazása

- A tételbizonyítás komplex feladat
 - n operátort tartalmazó tétel,
 - $O(2^n)$ hosszú bizonyítási szekvencia (közbenső tételek bizonyítása),
 - $O(2^{2^n})$ időigény a bizonyítás megtalálásához... (worst case, kereséssel)
 - Fontos a **bizonyítási stratégia meghatározása!**
- További használati esetek
 - Kézi bizonyítás automatikus ellenőrzése (**proof checking**)
 - Kézi bizonyítás segítése (**interaktív szabályalkalmazás**)
- Tételbizonyítás szerepe
 - Adat-orientált alkalmazások tulajdonságainak bizonyítása
 - Paraméter-függőségek kezelése (pl. protokoll résztvevők száma)
 - Indukció használható
 - Együttes használat a modellellenőrzéssel
 - Legkisebb paraméterértékre (kiindulási eset): **modellellenőrzés**
 - Tulajdonság megtartásának bizonyítása **indukcióval**
- Népszerű eszközök
 - HOL, PVS, ACL2, ...

Tételbizonyító rendszerek tulajdonságai

D tételbizonyító rendszer, **c** levezetendő (bizonyítandó) tétel

- Szemantikus helyesség (soundness):
 - Ami levezethető **D**-ben az igaz
 - Szükséges a használhatósághoz
 - Formálisan: $\forall c$: ha $\vdash_D c$ (levezethető) akkor $\models c$ (teljesül, igaz)
- Szemantikus teljesség (completeness):
 - Ami igaz, az levezethető **D**-ben
 - Hasznos tulajdonság, de nem szükséges (nem is mindig elérhető)
 - Formálisan: $\forall c$: ha $\models c$ akkor $\vdash_D c$
- Totális helyesség és teljesség:
 - Minden interpretációra helyes és teljes
- Konzisztens tételbizonyító rendszer:
 - Nem lehet egy tételt és az ellenkezőjét is bizonyítani

Kitekintés: Program és specifikáció kapcsolata

Egy P program és egy Φ specifikáció (tulajdonság) esetén:

- Verifikáció (program helyességbizonyítás):
 - $P \models \Phi$ eldöntése
- Analízis:
 - P alapján Φ levezetése
- Szintézis:
 - Φ alapján P konstruálása
- Optimalizálás:
 - $P \models \Phi$ alapján olyan P' konstruálása, hogy $P' \models \Phi$, ugyanakkor P' jobb adott szempontok szerint (kisebb, gyorsabb...)
- Javítás:
 - $P \not\models \Phi$ alapján olyan P' konstruálása, hogy $P' \models \Phi$ legyen

A verifikációs probléma leképezése

Források:

- Axiómákhoz:
 - Program utasítások (pl. értékadás, feltételkiértékelés)
 - Program domén axiómái (pl. integer, string, lista)
- Levezetési szabályokhoz:
 - Program nyelv szemantika
 - Program domén szemantika
- Bizonyítandó tételhez:
 - Program és a specifikáció
- Mi legyen a stratégia?

Bizonyításhoz leginkább használt stratégia: Indukció

- **Számítási indukció: Műveleti szemantika esetén**

- **Állapotokra:**

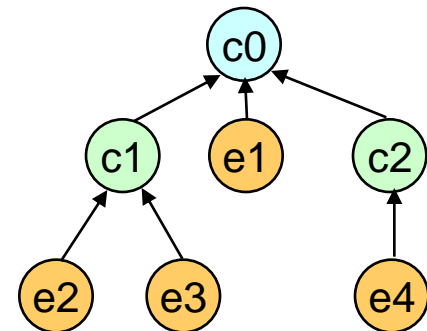
Ha a kiindulási állapot tulajdonságait ismerjük, az átmenetek követésével levezetjük a végállapot tulajdonságait



- **Strukturális indukció: Axiomatikus szemantikához**

- **Szintaktikai konstrukciókra:**

Ha az elemek tulajdonságait ismerjük, a rajtuk alkalmazott szintaktikai konstrukció alapján levezetjük a kompozit struktúra (így majd a teljes program) tulajdonságait



Célkitűzéseink

- **Bizonyítási vázak rögzítése program helyesség bizonyításhoz**
 - Bizonyítási **stratégia** megadása
 - Nincs teljesen automatikus „garantált” módszer
- **Nem konkrét programozási nyelvhez kötve**
 - Pseudo-nyelv (algoritmus leírás)
 - Továbbiakban mint „programozási nyelv” szerepel
 - Pl. saját, domén-specifikus nyelvhez is alkalmazható
- **Megkötések a bizonyíthatósághoz**
 - Programozási nyelv: **Formális szemantikával** rendelkezik (műveleti vagy axiomatikus szemantika)
 - Specifikációs nyelv: Elsőrendű kijelentéslogika

Programozási nyelv műveleti szemantikával

- Konfiguráció („állapot”): C
 - σ látható állapot (a program vizsgált kimenetében szerepel)
 - $\sigma[x]$ egy x változó értéke egy σ látható állapotban
 - $\sigma[\underline{x}]$ az \underline{x} változó-vektor értéke egy σ látható állapotban
 - Rejtett állapot (helyesség szempontjából érdektelen)
 - Szintaktikus folytatás λ : A további számításokat definiálja
 - „Programszámláló” helyett
 - Az adja meg, mit kell még végrehajtani (pl. forrásszöveggént)
- Átmenet reláció a konfigurációk között: \rightarrow
 - $\pi(P, \sigma_0)$ egy P program számítása σ_0 kezdőállapotból
 - $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \dots$ maximális szekvencia (végállapotig/végtelen)
 - $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \dots$ látható állapot szekvencia
 - $\text{val}(\pi(P, \sigma)) = \sigma_n$ véges esetben a végállapot jelölése
- Domén I: A számítások itt értelmezettek

Specifikációs nyelv

- Korlátozások a programra:
 - Determinisztikus
 - Nem folyamatos működésű (előbb-utóbb megáll)
 - Érték- vagy állapot-transzformációt valósít meg
- Tulajdonság megadása: Predikátumokkal
 - Előfeltétel: $p(\underline{x})$ - a megengedhető kezdeti állapotokra
 - \underline{x} a látható állapot változói
 - $\sigma_0 \models p(\underline{x})$ jelentése: kezdőállapotban igaz $p(\underline{x})$
 - Utófeltétel: $q(\underline{x})$ - az elfogadható végállapotokra
 - **true** – minden befejeződő számítás kielégíti
 - **false** – egy szabályos végállapot sem elégíti ki
 - $\text{val}(\pi(P, \sigma_0)) \models q(\underline{x})$ jelentése: π számítás végállapotában igaz $q(\underline{x})$
- Specifikáció példák
 - Segédváltozók használata
 - Specifikációs változók használata

Példák specifikációkra (integer domén)

- A program egy x és egy legalább ekkora y egészet ad:
Előfeltétel $p(x,y) = \text{true}$, utófeltétel $q(x,y) = x \leq y$
Tömörebben felírva $(\text{true}, x \leq y)$
- A program egy páros x egészet ad a kimenetén:
 $(\text{true}, \text{even}(x))$ ha a doménen értelmezett $\text{even}(x)$
 $(\text{true}, \exists y: x=2y)$ itt y kötött segédváltozó $q(x)$ -ben
- A program a bemeneti x -et megkétszerezi és kiadja:
 $(X=x, x=2X)$ itt X egy specifikációs változó
- A program az x/y pozitív bennfoglalás q eredményét és r maradékát adja:
 $(X=x \wedge x>0 \wedge Y=y \wedge y>0, X=qY+r \wedge 0 \leq r < Y)$
ha meg kell őrizni x és y értékét:
 $(X=x \wedge x>0 \wedge Y=y \wedge y>0, X=qY+r \wedge 0 \leq r < Y \wedge x=X \wedge y=Y)$

Program helyességi kritériumok (1)

- **Részleges helyesség: Jelölése** $\{p(\underline{x})\} P \{q(\underline{x})\}$

Egy P program részlegesen helyes $p(\underline{x})$, $q(\underline{x})$ szerint, ha teljesül a következő:

$\forall \pi(P, \sigma_0)$ és $\sigma_0 \models p(\underline{x})$ esetén

ha befejeződik π , akkor $\text{val}(\pi(P, \sigma_0)) \models q(\underline{x})$

- **Megjegyzések:**

- Az előfeltételt kielégítő állapotból induló számításokra: **ha befejeződik**, akkor az utófeltétel igaz a végállapotban
- Nem mond semmit azokról a számításokról, amelyekre $\sigma_0 \not\models p(\underline{x})$
- $\{\text{true}\}P\{\text{true}\}$ minden programra igaz
- $\{\text{true}\}P\{\text{false}\}$ ha igaz, akkor nincs befejeződő számítás

Program helyességi kritériumok (2)

- (Teljes) helyesség: Jelölése $\langle p(\underline{x}) \rangle P \langle q(\underline{x}) \rangle$

Egy P program helyes $p(\underline{x})$, $q(\underline{x})$ szerint, ha teljesül a következő:

$\forall \pi(P, \sigma_0)$ és $\sigma_0 \models p(\underline{x})$ esetén

π befejeződik és $\text{val}(\pi(P, \sigma_0)) \models q(\underline{x})$

- Megjegyzések:

- Az előfeltételt kielégítő állapotból induló számításokra: befejeződik és az utófeltétel igaz lesz a végállapotban

- $\langle p(\underline{x}) \rangle P \langle \text{true} \rangle$ csak befejeződést ír elő

- Felírható:

$\langle p(\underline{x}) \rangle P \langle q(\underline{x}) \rangle$ a.cs.a. $\{p(\underline{x})\} P \{q(\underline{x})\}$ és $\langle p(\underline{x}) \rangle P \langle \text{true} \rangle$

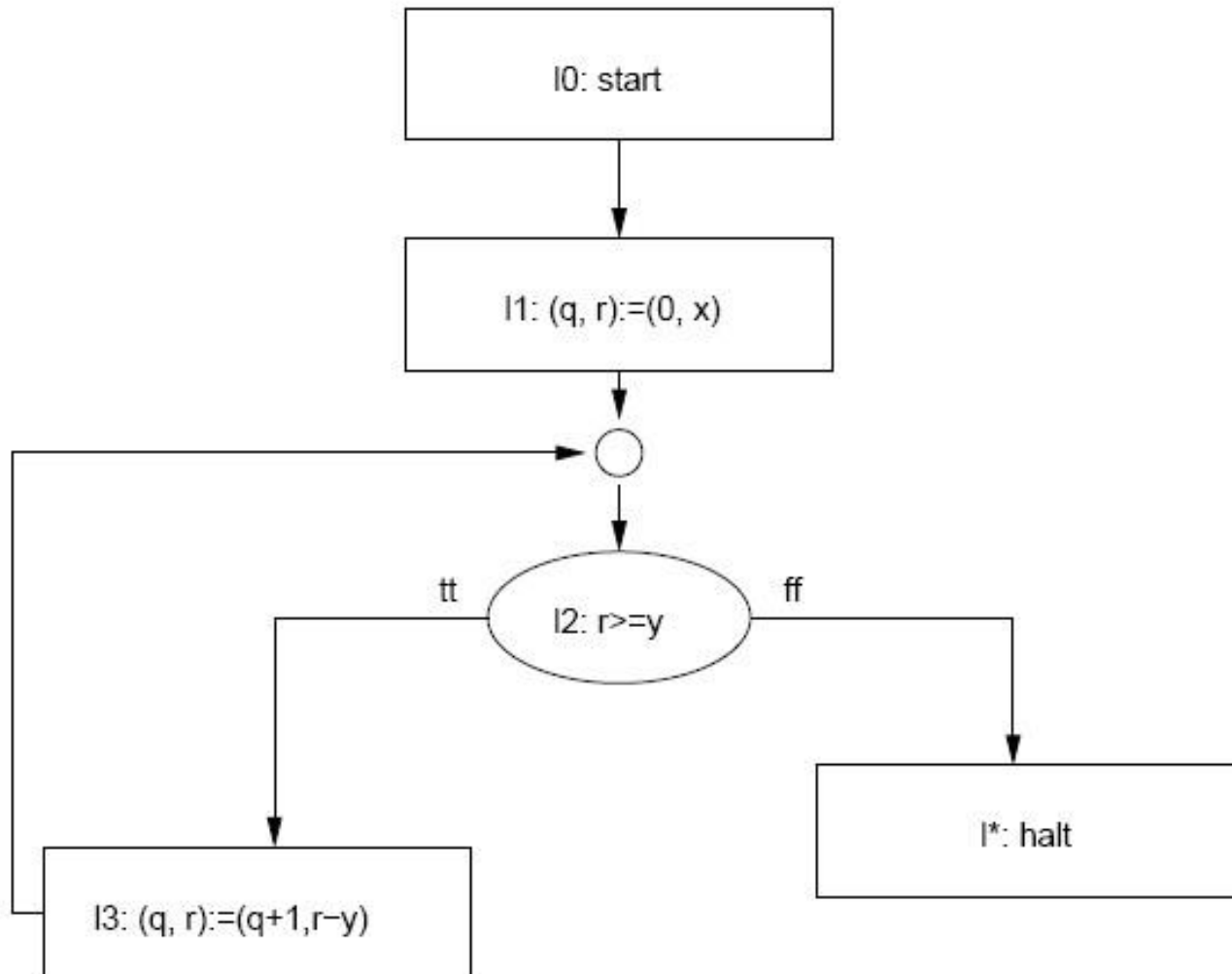
azaz a program helyes, ha részlegesen helyes és befejeződik

Egyszerű determinisztikus programok

- PLF „flow language”: Assemblyhez közeli pseudo-nyelv
 - $start, \underline{x}:=\underline{e}, B(\underline{x}), halt$ utasítások egyedi l címkékkel (l_0, l_*, l_1, \dots)
- Program felépítése: PLF mint véges irányított gráf
 - Csomópontok: utasítások; élek: utasítások sorrendje
 - Jelölés: $succ(l), succ^+(l), succ^-(l)$ adja meg a következő csomópontot
 - Minden utasítás egy $start \rightarrow halt$ útvonalon
- Szemantika: $C=(\sigma, \lambda)$ konfiguráció és \rightarrow megadása:
 $C(\sigma, \lambda) \rightarrow C'(\sigma', \lambda')$ a.cs.a.
 - λ egy $start$: $\lambda' = succ(\lambda), \quad \sigma' = \sigma$
 - λ egy $\underline{x}:=\underline{e}$ utasítás: $\lambda' = succ(\lambda), \quad \sigma' = \sigma[\underline{e}/\underline{x}]$
itt $[\underline{e}/\underline{x}]$ jelöli, hogy \underline{x} helyébe \underline{e} helyettesítése történik
 - λ egy $B(\underline{x})$ elágazás:
 - $\sigma \models B(\underline{x})$ esetén: $\lambda' = succ^+(\lambda), \quad \sigma' = \sigma$
 - $\sigma \not\models B(\underline{x})$ esetén: $\lambda' = succ^-(\lambda), \quad \sigma' = \sigma$

Példa: Maradékos osztás egész számokra

x/y számítása, q hányados, r maradék



Áttekintés a bizonyítási stratégiákról

- Részleges helyesség ciklusmentes programokra
 - Visszalépéses számítási indukció
- Részleges helyesség ciklust tartalmazó programokra
 - Induktív állítások módszere
- Teljes helyesség ciklust tartalmazó programokra
 - Befejeződés bizonyítása: Paraméterezett induktív állítások módszere

Részleges helyesség ciklusmentes esetben (1)

- Ötlet: Számítási indukció $\{p\}P\{q\}$ esetén
- Egy véges számításhoz tartozó u útvonal jellemzői:
 - $u = l_0 \rightarrow l_1 \rightarrow l_2 \rightarrow \dots \rightarrow l_k$
 - **Elérhetőségi (bejárás) feltétel:** $R_u(\underline{x})$ predikátum u útvonalhoz
 - Ha fennáll l_0 esetén, akkor éppen az u útvonalat járja be a program
 - **Állapot transzformáció:** $T_u(\underline{x})$ egy állapotvektort ad u útvonalhoz
 - \underline{x} állapotvektorból indulva az u útvonal bejárása utáni állapot $T_u(\underline{x})$
 - másként: $\underline{x} := T_u(\underline{x})$ a végállapotot eredményező állapot transzformáció
- Jelölések:
 - $l_m \rightarrow \dots \rightarrow l_k$ szuffixe az útvonalnak az m indextől (l_m pontból)
 - $R_u^m(\underline{x})$ és $T_u^m(\underline{x})$ erre a szuffixe vonatkoznak
- Ismert az útvonal l_k végpontjára (utolsó szuffixe):
 - $R_u^k(\underline{x}) = \text{true}$ - hiszen már a végpontban vagyunk
 - $T_u^k(\underline{x}) = \underline{x}$ - hiszen nincs további transzformáció

Részleges helyesség ciklusmentes esetben (2)

- **Visszalépéses indukció:**

- **Feltéve:** Ismert $R_u^{m+1}(\underline{x})$ és $T_u^{m+1}(\underline{x})$ egy szuffixre
- **Lépés:** Az I_m utasítása alapján $R_u^m(\underline{x})$ és $T_u^m(\underline{x})$ számítás

- $\underline{x} := \underline{e}$ hozzárendelés:

$$R_u^m(\underline{x}) = R_u^{m+1}(\underline{x})[\underline{e}/\underline{x}], \quad T_u^m(\underline{x}) = T_u^{m+1}(\underline{x})[\underline{e}/\underline{x}]$$

- $B(\underline{x})$ feltétel pozitív ága:

$$R_u^m(\underline{x}) = R_u^{m+1}(\underline{x}) \wedge B(\underline{x}), \quad T_u^m(\underline{x}) = T_u^{m+1}(\underline{x})$$

- $B(\underline{x})$ feltétel negatív ága:

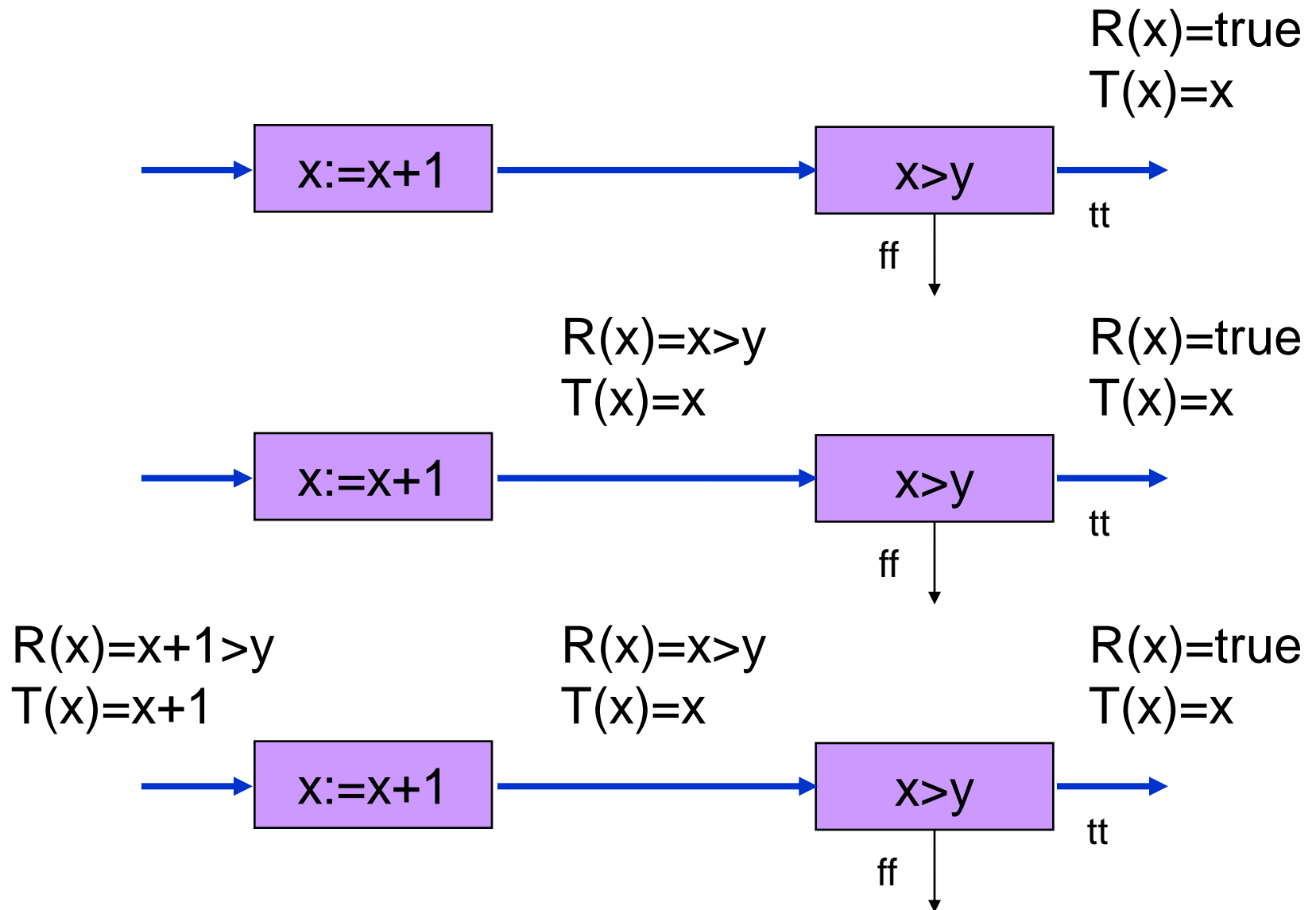
$$R_u^m(\underline{x}) = R_u^{m+1}(\underline{x}) \wedge \neg B(\underline{x}), \quad T_u^m(\underline{x}) = T_u^{m+1}(\underline{x})$$

- *start.*

$$R_u(\underline{x}) = R_u^0(\underline{x}), \quad T_u(\underline{x}) = T_u^0(\underline{x})$$

- Így a végpontban ismert $R_u^k(\underline{x}) = \text{true}$ és $T_u^k(\underline{x}) = \underline{x}$ alapján a kezdőpontra $R_u(\underline{x})$ és $T_u(\underline{x})$ *levezethető*

Részleges helyesség ciklusmentes esetben (példa)



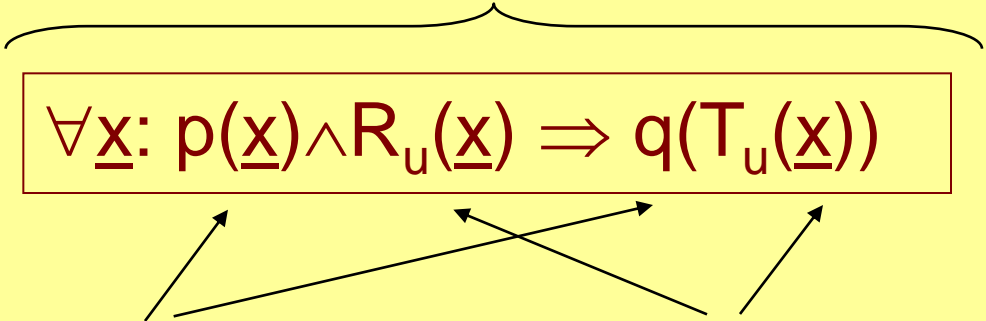
Részleges helyesség ciklusmentes esetben (3)

- Részleges helyesség megmutatása:

$\{p(\underline{x})\} P \{q(\underline{x})\}$ a.cs.a. ha minden teljes u útra:

$$\forall \underline{x}: p(\underline{x}) \wedge R_u(\underline{x}) \Rightarrow q(T_u(\underline{x}))$$

Doménen értelmezett elsőrendű logikai kifejezés;
ez tételbizonyítónak átadható minden u út esetén:


$$\forall \underline{x}: p(\underline{x}) \wedge R_u(\underline{x}) \Rightarrow q(T_u(\underline{x}))$$

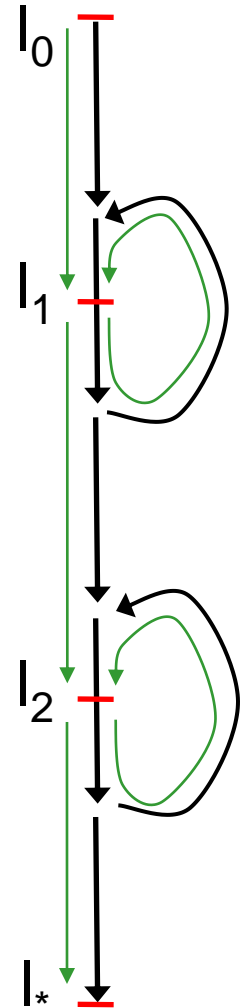
p és q a specifikáció alapján

R és T a program forrás alapján,
visszalépéses számítási indukció
alapján megadható minden útra

Részleges helyesség ciklusos programokra (1)

- **Ötlet: Ciklusok felvágása**

- Minden ciklusban egy I_i utasítás kijelölése, ami azt (ciklusmentes) szegmensekre osztja
- $I_{ij}(\underline{x})$ predikátum, ún. **induktív állítás** hozzárendelése a vágási ponthoz
 - Első belépés után I_i -nél igaz
 - Ciklus futása során igaz marad (**ciklus invariáns**)
 - Kilépés esetén következő szegmens bejárási feltételét igazzá teszi
→ majd az utófeltételt igazzá teszi a végpontban
- A szegmensek mint ciklusmentes utak az előző módszer alapján vizsgálhatók
 - Elérhetőségi feltétel és
 - állapot-transzformáció számítható



Részleges helyesség ciklusos programokra (2)

- Bizonyítási váz:

- Vágási pontok kijelölése a ciklusokban (legalább egy-egy)
- Induktív állítások felírása: $I_{li}(\underline{x})$
 - $I_{l_0}(\underline{x}) = p(\underline{x})$ - kezdőállapotra, vagy $p(\underline{x}) \Rightarrow I_{l_0}(\underline{x})$
 - $I_{l^*}(\underline{x}) = q(\underline{x})$ - végállapotra, vagy $I_{l^*}(\underline{x}) \Rightarrow q(\underline{x})$
 - Ciklusokban: ld. előbb (ciklus invariáns)
- Verifikációs feltételek: A szomszédos vágási pontokra, azaz minden l, l' vágási pontok által kijelölt u szegmensre:

$$\forall \underline{x}: I_l(\underline{x}) \wedge R_u(\underline{x}) \Rightarrow I_{l'}(T_u(\underline{x})) \text{ bizonyítandó}$$

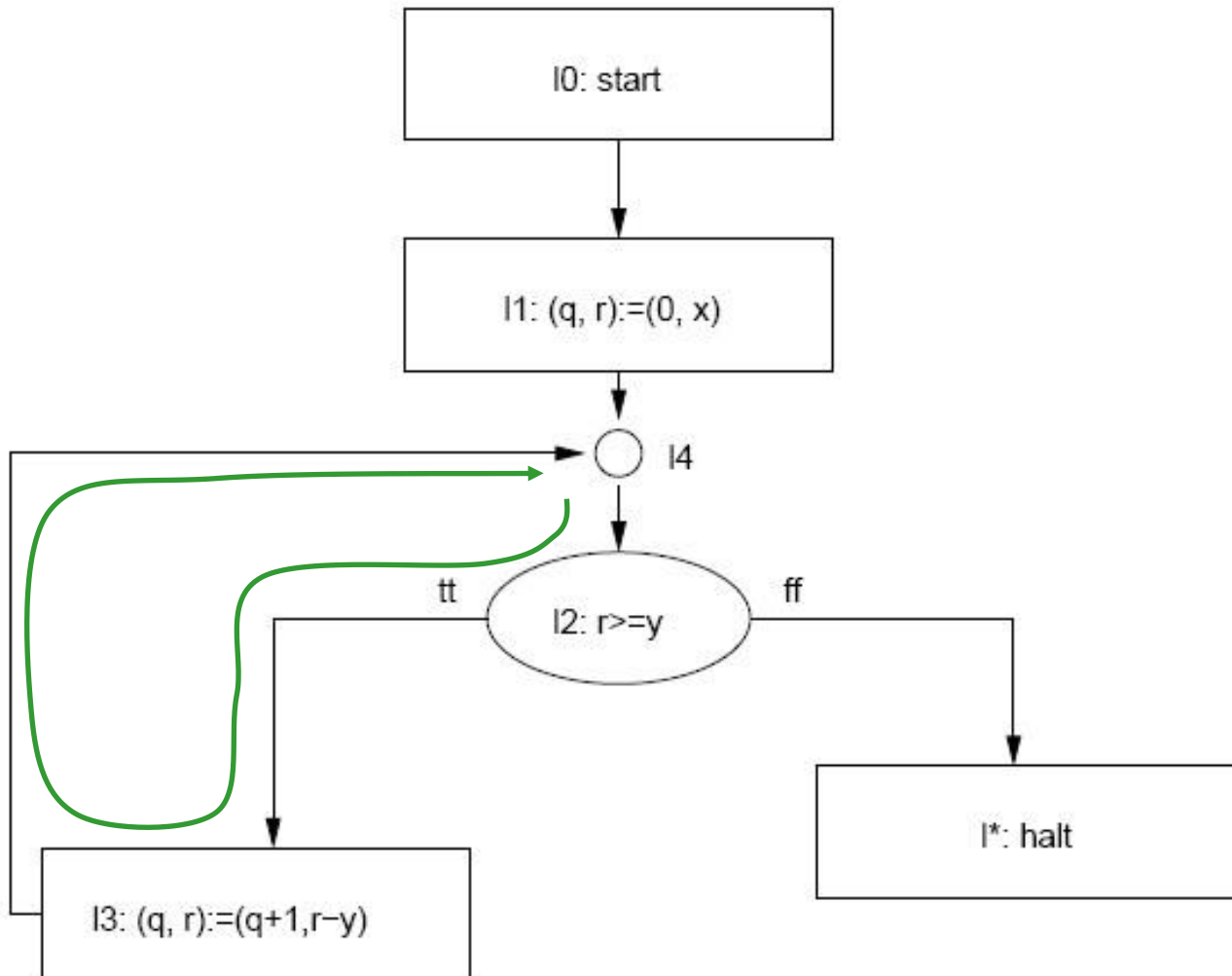
- $R_u(\underline{x})$ és $T_u(\underline{x})$ a szegmensekre kiszámíthatók

- Helyes és teljes módszer

- Mindig található megfelelő vágási pontok és induktív állítások (de ennek bizonyítása nem konstruktív ☹)
- Heurisztikus az induktív állítások felvétele

Részleges helyesség ciklusos programokra (példa)

$$I_{l_4}(x,y,q,r) = (x \geq 0 \wedge y > 0 \wedge x = q \cdot y + r \wedge r \geq 0)$$



Befejeződés bizonyítása ciklusok esetén (1)

- Ötlet: Az induktív állítások paraméterezése
 - Paraméter egy $(W, >)$ ún. jól megalapozott halmazból
 - Nem létezik végtelen $w_0 > w_1 > \dots$ csökkenő szekvencia, $w_i \in W$
 - Példák:
 - Természetes számok, és az ezeken értelmezett $>$ reláció
 - Véges halmaz valódi részhalmazai, és a tartalmazás reláció
 - Véges lista, és a lista prefix reláció
 - ...
 - A ciklus befejeződik, ha a paraméterről kimutatható, hogy **csökken** a ciklus végrehajtása során
 - A paraméter sok esetben lehet maga a ciklusváltozó, de segédváltozót is használhatunk
 - Megválasztása heurisztikus

Befejeződés bizonyítása ciklusok esetén (2)

- Bizonyítási váz:

- Jól megalapozott halmaz(ok) választása: $(W, <)$
- Vágási pontok kijelölése a ciklusokban: I_0, I_i, I_*
- Paraméterezett induktív állítások: $I_i(\underline{x}, w)$ ahol $w \in W$
- Verifikációs feltételek (bizonyítandók):
 - Inicializálás: $\forall \underline{x}: p(\underline{x}) \Rightarrow \exists w: I_{I_0}(\underline{x}, w)$ (előfeltétel bővítés)
 - Terminálás: $\forall \underline{x}: I_{I_*}(\underline{x}, w) \Rightarrow q(\underline{x})$
 - Csökkenés: Szomszédos I, I' pontok által kijelölt u szegmensre:

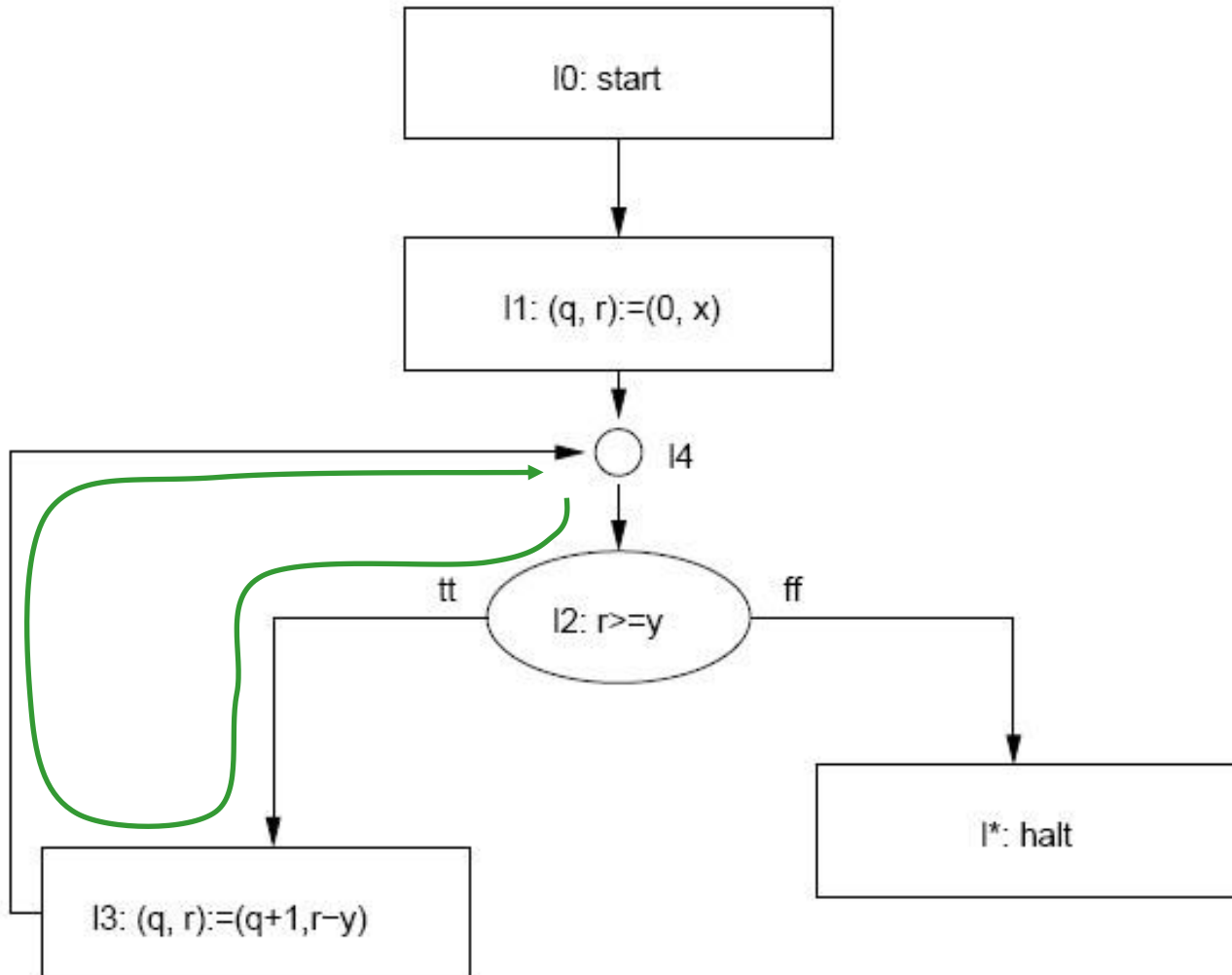
$$\forall \underline{x}: I_i(\underline{x}, w) \wedge R_u(\underline{x}) \Rightarrow \exists w' < w: I_{I'}(T_u(\underline{x}), w')$$

Itt $R_u(\underline{x})$ és $T_u(\underline{x})$ a szegmensekre kiszámíthatók

- Helyes módszer $\langle p(\underline{x}) \rangle P \langle \text{true} \rangle$ bizonyítására
 - A paraméterezett induktív állítások felvétele heurisztikus

Befejeződés bizonyítása ciklusok esetén (példa)

$$I_{l_4}(x, y, n) = (y > 0 \wedge n = r \geq 0)$$



Rész-összefoglaló

- Alacsony szintű pseudo-nyelvek:
 - Részleges helyesség ciklusmentes programokra:
 - Visszalépéses számítási indukció
 - Részleges helyesség ciklust tartalmazó programokra:
 - Induktív állítások módszere
 - Helyesség ciklust tartalmazó programokra:
 - Paraméterezett induktív állítások módszere,
jól megalapozott halmazból vett paraméterrel
- Következő lépés: Strukturált programnyelvek

Kitekintés: Szimbolikus végrehajtás

- **Eredeti ötlet**
 - King, „A new approach to program testing”, 1975
 - Szimbolikus megközelítés a konkrét változóértékek használata helyett
- **Módszer**
 - Programutak felmérése
 - Bejárás feltételek összeállítása az útvonal mentén a forráskódban (esetleg byte kódban) talált feltételek alapján
- **Alapelemek**
 - Szimbolikus (számítási) fa: Symbolic tree
 - Bejárás feltétel: Path constraint (PC)
 - „Programszámláló”: hol tart a végrehajtás

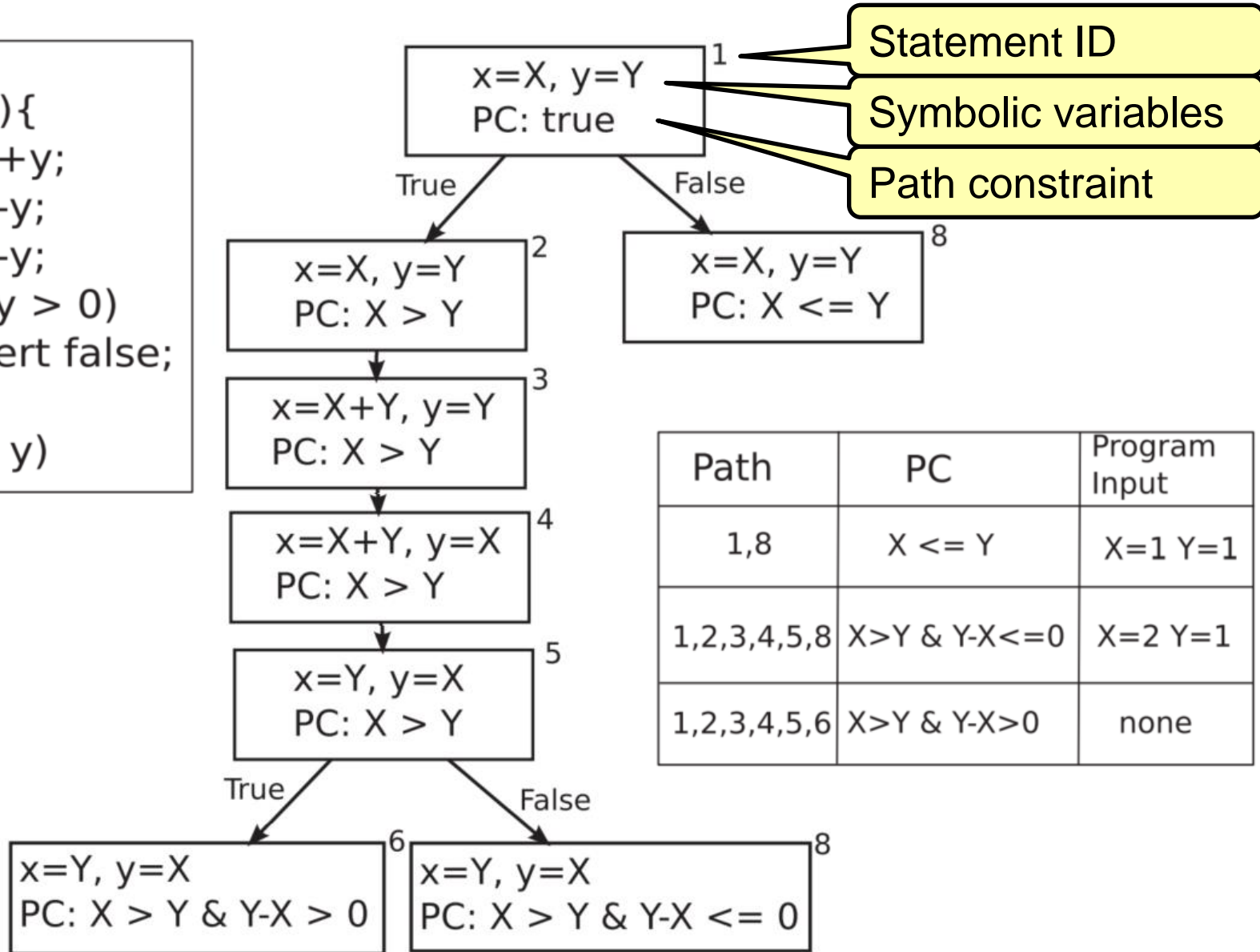
Kitekintés: Szimbolikus végrehajtás felhasználása

- Alkalmazás: Bemenetek generálása a forráskód alapján az egyes programutakhoz (teszteléshez, ld. később)
 - CSP vagy SMT megoldó használható
 - Nem teljesíthető bejárési feltétel: nem elérhető programút
 - Teljesíthető feltétel: futtatható programút
- Kihívások
 - Ciklusok: nagyszámú bejárható útvonal (korlátozott bejárás)
 - Bonyolult aritmetika támogatása az SMT eszközben
 - Külső függőségek (library-k) kezelése
- Eszközök
 - Java: Symbolic PathFinder (Java PathFinder alapokon)
 - .Net: PEX
 - C: KLEE, CUTE

Kitekintés: Szimbolikus végrehajtás, bejárési feltétel

```

int x, y;
1 if(x > y){
2   x = x+y;
3   y = x-y;
4   x = x-y;
5   if(x - y > 0)
6     assert false;
7 }
8 print(x, y)
    
```



Kitekintés: Szimbolikus végrehajtás, eszközök

Név	Platform	Hozzáférés	Megj. éve	Bemenet	Kimenet
CATG	Java	Nyílt forráskód	2012	Bájtkód	Tesztbemenet
CodePro AnalytiX	Java	Zárt forráskód	2001	Forráskód	JUnit tesztesetek
CREST	C	Nyílt forráskód	2008	Forráskód	Tesztesetek
CUTE & jCUTE	C & Java	Nem elérhető	2005	Forráskód/bájtkód	Tesztbemenet
DART	C	Nem elérhető	2005	Forráskód	?
EXE	C	Nem elérhető	2006	Forráskód	?
LCT	Java	Nyílt forráskód	2010	Bbájtkód	Tesztbemenet
KLEE	C	Nyílt forráskód	2008	LLVM bájtkód	Tesztbemenet
Palus	Java	Nyílt forráskód	2010	Bájtkód	Tesztesetek
PET/jPET	Java	Nyílt forráskód	2009	Forráskód	Tesztesetek
PEX	.NET	Zárt forráskód	2008	Bájtkód	Tesztesetek
SPF	Java	Nyílt forráskód	2012	Bájtkód	Tesztbemenet

Forrás: Cseppentő Lajos, „Szimbolikus végrehajtást használó tesztgeneráló eszközök egységes összehasonlítása”, kari TDK, 2013

Helyességbizonyítás strukturált programokra

- Cél a „komponálhatóság”:
 - Ha egy P program P_1 és P_2 szintaktikai egységekből áll, akkor P tulajdonságai P_1 és P_2 tulajdonságai alapján bizonyíthatók
 - Strukturális indukció elve

- Strukturált programok: PLW nyelv

$P ::= x := e \mid \text{skip} \mid P_1; P_2 \mid \text{if } B \text{ then } P_1 \text{ else } P_2 \text{ fi} \mid \text{while } B \text{ do } P \text{ od}$

- Példa (maradékös osztás):

$P_{\text{div}}: r := x; q := 0; \text{while } r \geq y \text{ do } r := r - y; q := q + 1 \text{ od}$

PLW műveleti szemantikája

- Konfiguráció: $C_i = (P_i, \sigma_i)$ ahol
 - P_i a szintaktikus folytatás (E az üres folytatás jelölése)
 - σ_i a látható állapot (állapotváltozók)
- Átmenet reláció: $C \rightarrow C'$
 - $(x := e, \sigma) \rightarrow (E, \sigma[e/x])$
 - $(\text{skip}, \sigma) \rightarrow (E, \sigma)$
 - $(P_1; P_2, \sigma) \rightarrow (P_1'; P_2, \sigma')$ ha $(P_1, \sigma) \rightarrow (P_1', \sigma')$
 - $(\text{if } B \text{ then } P_1 \text{ else } P_2 \text{ fi}, \sigma) \rightarrow (P_1, \sigma)$ ha $\sigma[B] = \text{true}$
 $\rightarrow (P_2, \sigma)$ ha $\sigma[B] = \text{false}$
 - $(\text{while } B \text{ do } P \text{ od}, \sigma) \rightarrow (P; \text{while } B \text{ do } P \text{ od}, \sigma)$ ha $\sigma[B] = \text{true}$
 $\rightarrow (E, \sigma)$ ha $\sigma[B] = \text{false}$

Itt az $E;P \equiv P$
azonosság
alkalmazható
a végén

D dedukciós rendszer részleges helyesség bizonyításához (1)

- Axiómák:

- ASS: $\{p[e/x]\} x:=e \{p\}$

Utófeltételként p teljesül, ha előfeltételként $p[e/x]$ teljesül

- SKIP: $\{p\} \text{ skip } \{p\}$

- Szabályok a szintaktikai struktúrákhoz:

- SEQ:
$$\frac{\{p\} P_1 \{r\} \text{ és } \{r\} P_2 \{q\}}{\{p\} P_1; P_2 \{q\}}$$

Ha (feltétel)
akkor (köv.)

Segédtételek
Biz. tétel

- COND:
$$\frac{\{p \wedge B\} P_1 \{q\} \text{ és } \{p \wedge \neg B\} P_2 \{q\}}{\{p\} \text{ if } B \text{ then } P_1 \text{ else } P_2 \text{ fi } \{q\}}$$

- REP:
$$\frac{\{p \wedge B\} P \{p\}}{\{p\} \text{ while } B \text{ do } P \text{ od } \{p \wedge \neg B\}}$$

p ciklus
invariáns

D dedukciós rendszer részleges helyesség bizonyításához (2)

- Általános szabályok:

- CONS:
$$\frac{p \Rightarrow p_1 \text{ és } \{p_1\} P \{q_1\} \text{ és } q_1 \Rightarrow q}{\{p\} P \{q\}}$$

Előfeltétel
bővítés és
utófeltétel
szűkítés

- AND:
$$\frac{\{p\} P \{q_1\} \text{ és } \{p\} P \{q_2\}}{\{p\} P \{q_1 \wedge q_2\}}$$

Utófeltétel
részekre
bontása

- OR:
$$\frac{\{p_1\} P \{q\} \text{ és } \{p_2\} P \{q\}}{\{p_1 \vee p_2\} P \{q\}}$$

Előfeltétel
szétválasztása
esetekre

- Domén axiómái és szabályai:

A tételbizonyítónak szintén ismernie kell

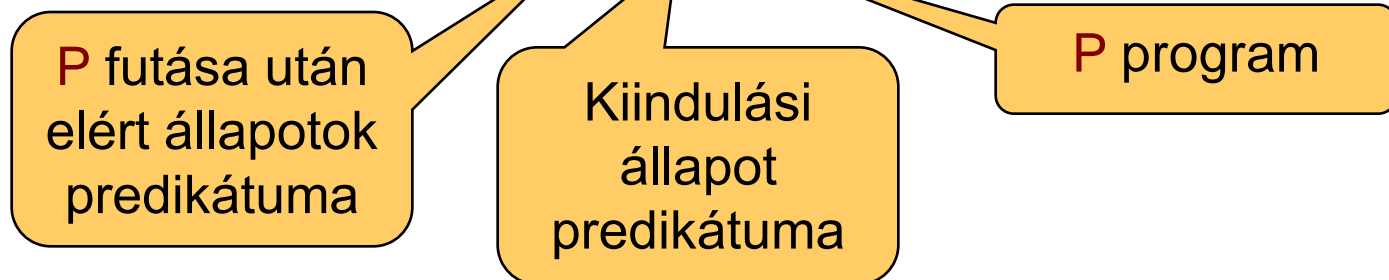
D dedukciós rendszer részleges helyesség bizonyításához (3)

- Egy C állítás bizonyítása: $Tr_1 \vdash_D C$ ahol
 - I a domén, Tr_1 a domén axiómái és szabályai
 - D a dedukciós rendszer
- Jellemzők:
 - Az előbb felvázolt D helyessége bizonyítható
 - $Tr_1 \vdash_D \{p\}P\{q\}$ következménye $\models \{p\}P\{q\}$
 - D teljessége:
 - Ha a domén axióma- és szabályrendszere kellően bonyolult (elegendően erős): **Gödel első nemteljességi tétele** érvényes, azaz lehet olyan igaz állítás, ami nem bizonyítható
- Gyakorlati problémák:
 - A programnyelv szemantikáját le kell képezni szabályokra és axiómákra
 - A tételbizonyítónak ismernie kell a domén szabályait is
 - Szabályok alkalmazásához stratégia (vagy keresés) kell
 - Specifikációs nyelv kifejezőképessége elegendő-e?

Specifikációs nyelv kifejezőképessége

- Definíciók:

- SL specifikációs nyelv kifejező egy PL programnyelv és I domén esetén, ha $\forall p \in SL, \forall P \in PL$ esetén $post_1(p, P)$ kifejezhető SL-ben



- Egy D dedukciós rendszer relatív teljes a részleges helyesség bizonyításához, ha $\forall SL, \forall PL, \forall I$ esetén, ahol SL kifejező PL-re és I-re, fennáll: $\models_I \{p\}P\{q\}$ következménye $Tr_1 \vdash_D \{p\}P\{q\}$

D* dedukciós rendszer helyesség bizonyításhoz

- Cél: PLW programok helyességének bizonyítása
 - `while B do P od` ciklusok befejeződése kérdéses
- Ötlet (itt is): Paraméterezett állítások
 - Jól megalapozott halmaz (itt pl. `n` természetes szám)
 - $pi(\underline{x},n)$ ciklus invariáns választása kell
- Módosuló REP szabály ebben az esetben:

- REP*:
$$\frac{pi(\underline{x},n) \Rightarrow B \text{ és } \langle pi(\underline{x},n) \rangle P \langle pi(\underline{x},n-1) \rangle \text{ és } pi(\underline{x},0) \Rightarrow \neg B}{\langle \exists n: pi(\underline{x},n) \rangle \text{ while } B \text{ do } P \text{ od } \langle pi(\underline{x},0) \rangle}$$

- Többi szabály:
{...} helyett egyszerűen $\langle \dots \rangle$ kell

Aritmetikai kiterjesztés

- A módosult REP* szabály miatt:
 - SL-nek támogatnia kell a jól megalapozott halmaz használatát (itt: természetes számok)
- Tipikus eset: Peano-aritmetika támogatása
 - Természetes számok és $+$, $*$, $s()$ műveletek
 - Tételbizonyító erre felkészítve
- Definíciók:
 - SL aritmetikai kiterjesztése SL^+ , ha minimális bővítésként a Peano-aritmetikát tartalmazza
 - I domén aritmetikai kiterjesztése I^+ , ha minimális bővítésként a Peano-aritmetika doménjét tartalmazza

Aritmetikai helyesség és teljesség

- Aritmetikai helyesség $p, q \in SL^+$ mellett:
 - $Tr_{I_+} \Vdash_D \langle p \rangle P \langle q \rangle$ következménye $\models_{I_+} \langle p \rangle P \langle q \rangle$
- Aritmetikai teljesség $p, q \in SL^+$ mellett:
 - $\models_{I_+} \langle p \rangle P \langle q \rangle$ következménye $Tr_{I_+} \Vdash_D \langle p \rangle P \langle q \rangle$
- Itt: D^* aritmetikai helyessége bizonyítható

Összefoglalás

- Alacsony szintű pseudo-nyelvek:
 - Részleges helyesség ciklusmentes programokra:
 - Visszalépéses számítási indukció
 - Részleges helyesség ciklust tartalmazó programokra:
 - Induktív állítások módszere
 - Helyesség ciklust tartalmazó programokra:
 - Paraméterezett induktív állítások módszere
- Strukturált programnyelvek (while programok):
 - Részleges helyesség:
 - Dedukciós rendszer (strukturális indukció)
 - Helyesség:
 - Dedukciós rendszer paraméterezett állításokkal
 - Aritmetikai kiterjesztés, aritmetikai helyesség és teljesség

Program helyességbizonyítás a gyakorlatban

Néhány példa:

- **Spec# Programming System: C# kiterjesztés**
 - Előfeltételek, utófeltételek megadása (metódusokhoz)
 - Objektum invariánsok (pl. adattartományok)
 - Boogie: Az utófeltételek automatikus ellenőrzése
- **JML: Java Modelling Language**
 - Előfeltételek, utófeltételek, invariánsok megadása
 - ESC/Java2: JML részhalmazhoz automatikus utófeltétel bizonyítás
- **SPARK: Ada nyelvi részhalmaz**
 - Interaktív tételbizonyítóval támogatott ellenőrzés
- **B módszer: Speciális modellezési nyelv és megközelítés**
 - Bizonyítandó tételek származtatása és a tételbizonyítás nagy részben automatikus
 - Fókusz: Specifikáció konzisztens finomítása

Eszköz példa: Spec# és Boogie2

Spec# Programming System
Microsoft

Ld: Dmitriy Dunaev referátuma

Spec# részletek

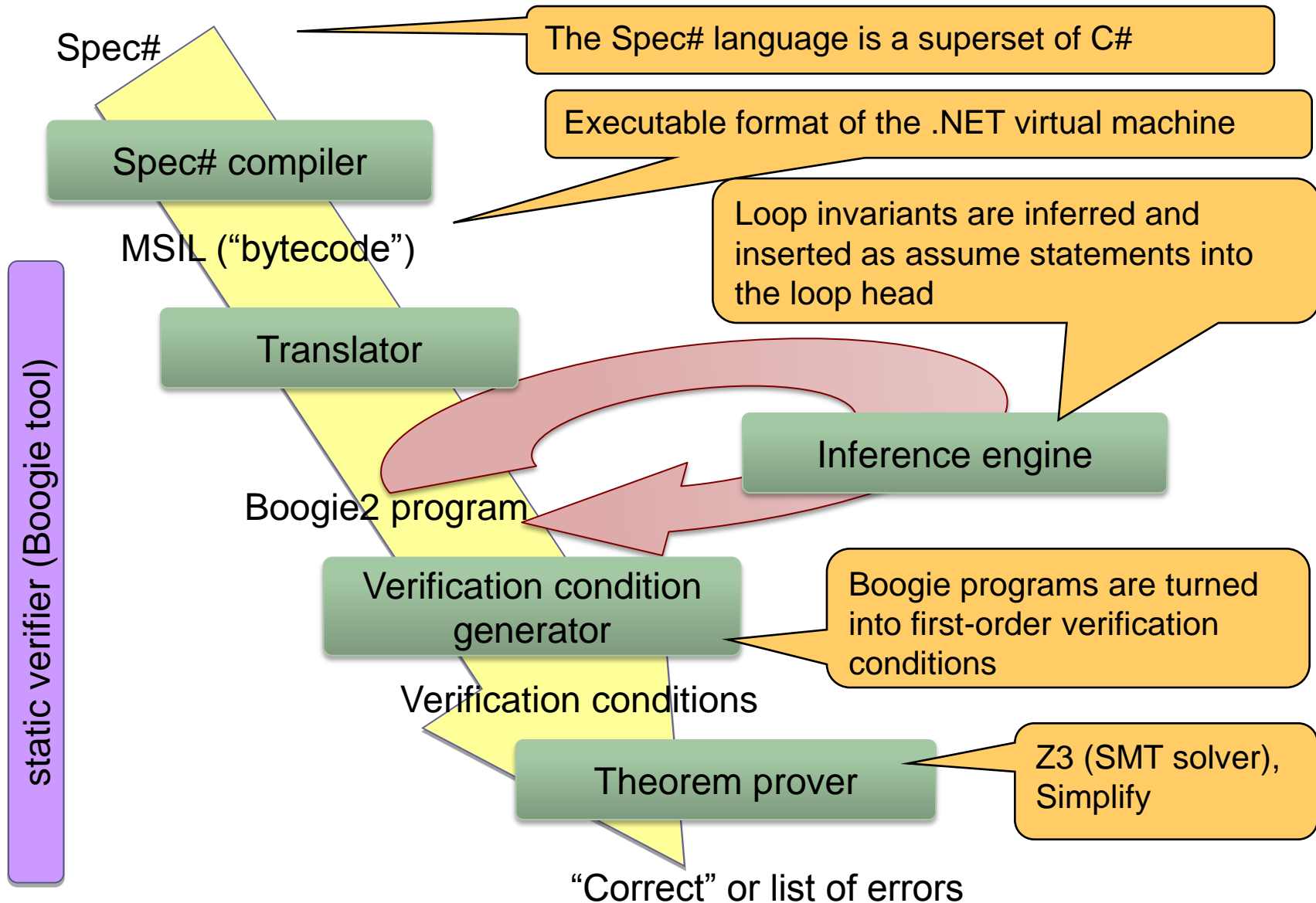
```
int day_of_week;  
  
public void newDOW(int day)  
  requires (day <= 7);  
  requires (day >= 1);  
  {  
    day_of_week = day;  
  }
```

```
class Meeting {  
  int day_of_week;  
  invariant (1 <= day_of_week);  
  invariant (7 > day_of_week);  
  
  void newDOW(int day )  
  {  
    day_of_week = day;  
  }  
}
```

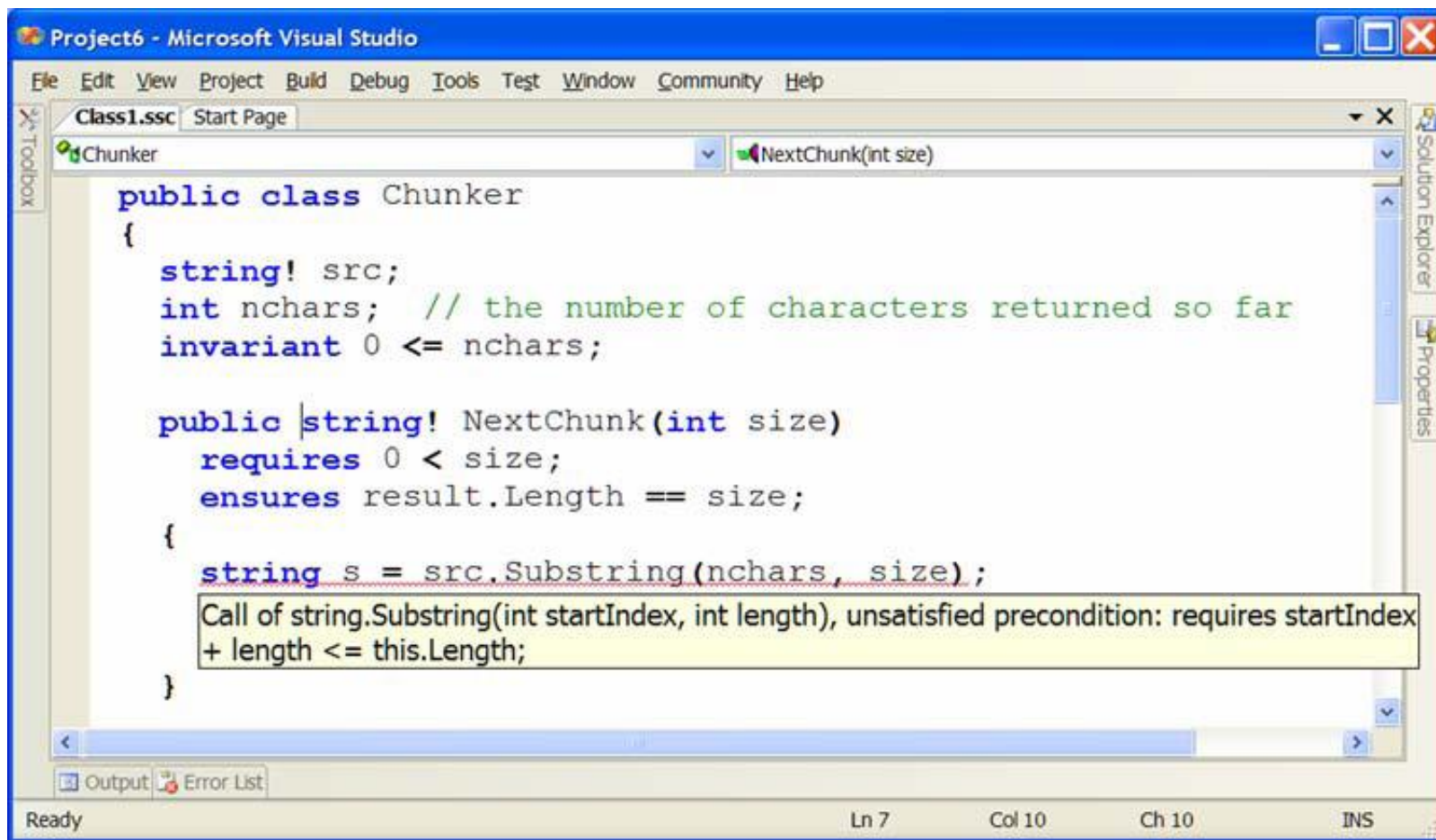
```
static void Swap(int[] a, int i, int j)  
  requires 0 <= i && i < a.Length;  
  requires 0 <= j && j < a.Length;  
  modifies a[i], a[j];  
  ensures a[i] == old(a[j]);  
  ensures a[j] == old(a[i]);  
  {  
    int temp;  
    temp = a[i];  
    a[i] = a[j];  
    a[j] = temp;  
  }
```

- Formal language for API contracts (influenced by JML, AsmL, Eiffel)
- Extends C# with constructs for preconditions, postconditions, invariants

Működés



Eszköztámogatás



- Telepítés: .NET, Visual Studio, Spec# compiler, Boogie2, Z3, Simplify