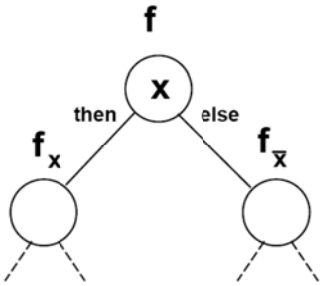
FMOS teorija – sažetak 3. ciklusa
Simbolički postupci i ROBDD

- **Simbolički postupci verifikacije sustava provjerom modela**
  - Eksplozija stanja čini skupove stanja neprikladnim za eksplicitnu manipulaciju u memoriji računala
  - Simbolički postupci temelje se na uporabi logičkih (Booleovih) funkcija
    - manipulariju sa skupovima stanja, a ne individualnim stanjima
    - smanjuju zahtjeve na prostorne i vremenske resurse tijekom verifikacije
  - skupovi (relacije) kodiraju se Booleovim funkcijama
  - BDD dijagrami su prikladna metoda predstavljanja Booleovih funkcija
  - **Predstavljanje logičkih funkcija tablicom**
    - prednosti: kanonski, razumljiv, jednostavna provjera ekvivalencije
    - nedostaci: eksponencijalni prostor (za n varijabli 2^n redaka u tablici)
  - **Predstavljanje logičkih funkcija mintermima i makstermima**
    - prednosti: kanonski, razumljiv, jednostavna provjera ekvivalencije
    - nedostaci: nije minimalni oblik funkcije
  - **Predstavljanje logičkih funkcija u standardnom (dvorazinskom) obliku**
    - SOP (suma produkata) i POS (produkt suma)
    - prednosti: kompaktan opis
    - nedostaci: nije kanonski, računalno vrlo skupa pretvorba SOP-POS, skupo određivanje ekvivalencije, zadovoljivost POS-a je NP kompletno (3-SAT)
  - **Predstavljanje logičkih funkcija u nestandardnom (višerazinskom) obliku**
    - nije POS nego PSOP
    - prednosti: kompaktan opis
    - nedostaci: nije kanonski, računalno izrazito skupa pretvorba u standardne oblike, skupo određivanje ekvivalencije i zadovoljivosti
  - **Predstavljanje logičkih funkcija Booleovim kockama**
    - prednost: vizualno razumljivo predstavljanje logičkih funkcija i temeljnih teorema i pravila
    - nedostatak: za n>4 predstavljanje neprimjereno
- **Shannonov teorem ekspanzije**
  - Svaka logička funkcija može se ekspandirati oko varijable x_i u SOP oblik:
    Varijabla x_i naziva se varijabla cijepanja (*splitting*).
    Pri tome su f_x i $f_{x'}$ kofaktori.

$$f = x_i f_{x_i} + \overline{x_i} f_{\overline{x}_i}$$

  - 

- **BDD (Binary Decision Diagram)**
  - usmjereni, aciklički graf G(V, E)
  - binarni – svaki čvor (osim terminalnih) ima točno dvoje djece
  - kanonski (jedinstven) – za svaku Booleovu funkciju s konačnim brojem varijabli možemo napraviti jedinstveni BDD
    - ne postoji čvor s jednakom djecom, tj. $then(v) = else(v)$
    - ne postoje izomorfni podgrafovi (podgrafovi s istom strukturom i označavanjem)
  - ***ite*** **– if-then-else**
    - $ite(F, G, H) = if\ F\ then\ G\ else\ H = FG + F'H$
Name:	Notation:	ITE form:
not	$\overline{F}$	ITE(F, 0, 1)
and	$F \cdot G$	ITE(F, G, 0)
xor	$F \oplus G$	ITE(F, $\overline{G}$, G)
or	$F \vee G$	ITE(F, 1, G)
nor	$\overline{F \vee G}$	ITE(F, 0, $\overline{G}$)
equiv	$F \leftrightarrow G$	ITE(F, G, $\overline{G}$)
implies	$F \rightarrow G$	ITE(F, G, 1)
nand	$\overline{F \cdot G}$	ITE(F, $\overline{G}$, 1)
    - ako *ite* operator ima kao prvi parametar samostalnu varijablu, tada *ite* predstavlja BDD s čvorom te varijable
    - $$\begin{aligned} ite(f,g,h) &= fg + \overline{f}h \\ &= v(fg + \overline{f}h)_v + \overline{v}(fg + \overline{f}h)_{\overline{v}} \\ &= v(f_v g_v + \overline{f}_v h_v) + \overline{v}(f_{\overline{v}} g_{\overline{v}} + \overline{f}_{\overline{v}} h_{\overline{v}}) \\ &= ite(v, ite(f_v, g_v, h_v), ite(f_{\overline{v}}, g_{\overline{v}}, h_{\overline{v}})) \\ &= (v, ite(f_v, g_v, h_v), ite(f_{\overline{v}}, g_{\overline{v}}, h_{\overline{v}})) \end{aligned}$$
    - $$\begin{aligned} ite(F,F,G) &\Longrightarrow ite(F,1,G) \\ ite(F,G,F) &\Longrightarrow ite(F,G,0) \\ ite(F,G,\overline{F}) &\Longrightarrow ite(F,G,1) \\ ite(F,\overline{F},G) &\Longrightarrow ite(F,0,G) \end{aligned}$$
    - $$\begin{aligned} ite(F,1,G) &\equiv ite(G,1,F) \\ ite(F,0,G) &\equiv ite(\overline{G},0,\overline{F}) \\ ite(F,G,0) &\equiv ite(G,F,0) \\ ite(F,G,1) &\equiv ite(\overline{G},\overline{F},1) \\ ite(F,G,\overline{G}) &\equiv ite(G,F,\overline{F}) \end{aligned}$$
- **ROBDD (Reduced Ordered Binary Decision Diagram)**
  - ROBDD ne numerira eksplicitno putove (kao tablica), već graf predstavlja eksponencijalan broj putova s linearnim brojem čvorova
  - svaki čvor u ROBDD-u predstavlja korijenski čvor ROBDD-a jedne jedinstvene logičke funkcije
  - **jedinstvene tablice (*unique table*)**
    - zapisuje pokazivače na čvorove ROBDD-a
    - njome se ostvaruje kanonski zapis, jer za neki čvor postoji jedan i samo jedan indeks (pokazivač), a to je adresa strukture opisa tog čvora u jedinstvenoj tablici
    - odgovara na pitanje *„Da li postoji jedinstveni čvor (v, g, h)?“*
    - za efiksaniju pohranu u memoriji računala na indekse vršnih varijabli i na indekse njihovih odgovarajućih *then* i *else* strana primjenjuje se *hash* funkcija što rezultira u stvarnoj adresi jedinstvene tablice
      - u slučaju *hash* kolizija, formira se kolizijski lanac na *hash* indeksu
  - **izračunske tablice (*computed table*)**
    - zapisuje pokazivače na rezultate *ite* funkcije
    - odgovara na pitanje „Da li smo već izračunali nešto?“ za izbjegavanje ponovnog računanja ROBDD strukture koja predstavlja traženi rezultat

- rezultat izračunavanja je konačno jedan ROBDD pa izračunska tablica pohranjuje argumente izračunavanja i pokazivač na jednaku strukturu podataka kao i jedinstvena tablica, tj. na čvor (v, g, h)
- izračunska tablica ne koristi kolizijski lanac (kod kolizije stariji podatak se odbacuje)
- bez upotrebe izračunske tablice
  - jedan pristup jedinstvenoj tablici (bez rekurzije)
  - rekurzivni poziv (ako nije završni) traži 2 nova pristupa tablici
  - vrijeme izvođenja je eksponencijalno prema broju varijabli
- uz uporabu izračunske tablice
  - za jedinstvenu (f, g, h) funkcija $ite(f, g, h)$ se poziva jednom
  - ($|f|$ - broj čvorova) općenito $ite(f, g, h)$ se poziva $|f| \times |g| \times |h|$ puta, pa je složenost $O(|f| \times |g| \times |h|)$

- ***ite_constant*(f, g, h)**
  - provjera da li je *ite* funkcija konstanta; vraća 0, 1 ili NC (nije konstanta)

```
ITE_constant(F, G, H){
    if (trivial case) {
        return result (0,1, or NC);
    } else if (cache table has entry for (F, G, H))
        return result;
    } else {
```

  - npr. za $ite(F, G, 1)$ odmah se vraća NC ako H≠1
  - ako je ranije izračunato vrati rezultat, a ako ne idi na postupak prikazan na sljedećoj slici
  - da bi funkcija bila konstanta, THEN i ELSE strana u svakoj iteraciji moraju biti jednake konstante
    - algoritam *ite_ constant* je brži od standardnog **samo ako funkcija nije konstanta** (raniji izlazak)
- **Proširenje oznakama komplementa**
  - uvijek se koristi lijevi graf, tj. onaj koji nema oznaku komplementa na luku *then* strane; eventualni komplement je na *else* (0) strani
  - u ekvivalenciji trojki treba koristiti prvi lijevi zapis

$$ite(F, G, H) \equiv ite(\overline{F}, H, G) \equiv \overline{ite(F, \overline{G}, \overline{H})}, \equiv \overline{ite(\overline{F}, \overline{H}, G)}$$

- **Upravljanje memorijom**
  - za neku ROBDD strukturu postoje pokazivači na čvorove u memoriji; mnogi od tih pokazivača referenciraju isti čvor ROBDD strukture
  - upravljanje memorijom mora voditi računa da izbriše čvor ako ne postoji niti jedan pokazivač na njega
  - efikasno upravljanje memorijom traži da se struktura podataka svakog čvora proširi s podatkom o referenciji na taj čvor
- **Nedostaci ROBDD-a**
  - za mnoge logičke funkcije veličina ROBDD-a (broj čvorova) je polinomska u odnosu na broj varijabli, ali **samo za dobru uređenost varijabli**
  - za neke logičke funkcije, veličina je eksponencijalna u odnosu na broj varijabli, bez obzira na uređenost
  - korisna heuristika: uzmi varijablu koja je najzastupljenija u logičkom izrazu

- **Primjena ROBDD-a u formalnoj verifikaciji sustava**

Set	BDD
$\varnothing$	BDD_0
$\overline{S}$	bdd_not(S)
$S \cup T$	bdd_or(S,T)
$S \cap T$	bdd_and(S,T)
S = T	S=T
Universe	BDD_1

ite izračun
0
ite(S, 0, 1)
ite(S, 1, T)
ite(S, T, 0)
ite(S, T, T')
1

- sve uporabljene funkcije mogu se računate *ite* operatorom
  - bdd_equiv(f, g)= *ite(f, g, g')*
  - bdd_and(f, g) = *ite(f, g, 0)*
  - bdd_exist(var, f) = *ite*(f_{var} , *1*, $f_{var'}$)
- **Izračun CTL formule: EX p**
  - Izračun ROBBD-ovima

```
BDD EX (BDD p)
{
      return H^-1(p);
}
```

  - Kako je H: $\{s\} \rightarrow \{t\}, to\ H-1: \{t\} \rightarrow \{s\}$. Operacija se u ROBDD-u postiže jednostavnom zamjenom s_i sa t_i u ranije prikazanom algoritmu za H. To je moguće jednim prolazom kroz ROBDD ako je redoslijed varijabli: s_1, t_1, s_2, t_2, ...
- **Izračun CTL formule: EG p**
  - Izračun skupovima: Q(EG p) = Q(p) ∩ Q(EX EG p)
    Q(H^{-1} (EG p))
  - Izračun ROBDD-ovima:

```
BDD EG (BDD p) {
      k:=1;  Z_k:=p;
        do  {
            Z_k+1:= bdd_and(p, H^-1(Z_k));
            if (Z_k+1 = Z_k) return Z_k ;
            k++;
            } forever; }
```

- **Izračun CTL formule: E(p U q) = (p EU q)**
  - Izračun skupovima: Q(p EU q)) = Q(q) ∪ Q(p) ∩ Q(EX (p EU q))
  - Izračun ROBDD-ovima

```
BDD EU (BDD p, BDD q) {
      k:=1;  Z_k:=q;
        do  {
            Z_k+1:= bdd_or(q, bdd_and(p, H^-1(Z_k));
            if (Z_k+1 = Z_k) return Z_k ;
            k++;
            } forever; }
```