BDD synthese

Rick van der Zwet
<hvdzwet@liacs.nl>

4 augustus 2010

1 Inleiding

De hierin besproken inhoud is een samenvatting gemaakt van pagina 86–88 en de daarin genoemde opgave 60 en 63 uit het college boek [DK2009] naar aanleiding van het vak Seminar Combinatorial Algorithms 2010 [SCA2010].

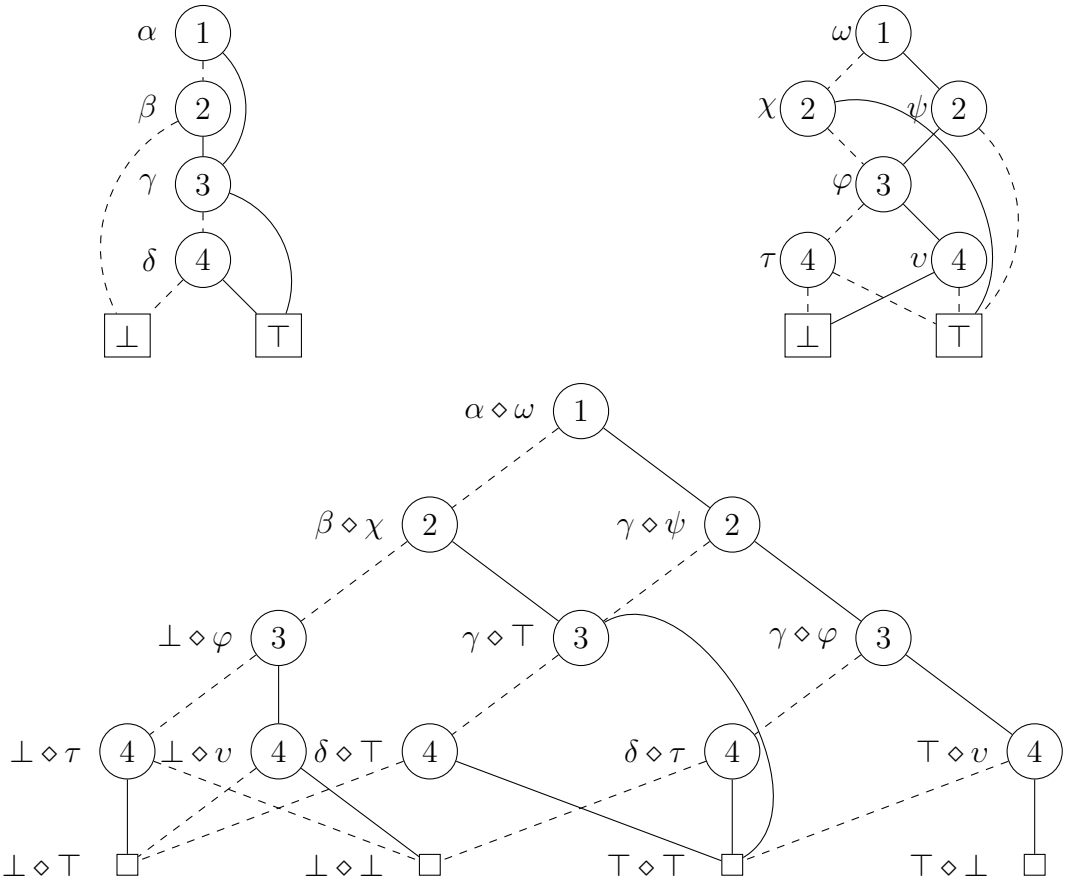
2 Introductie BDD

Synthese van een Binary Decision Diagram *BDD* is een belangrijk *BDD* algoritme [DK2009, p. 86]. Het neemt in essentie een *BDD* functie f en combineert deze met een andere *BDD* functie g zodanig dat er een nieuwe *BDD* ontstaat voor een nieuwe functie, bijvoorbeeld $f \wedge g$ of $f \oplus g$. De reden dat dit zo belangrijk is komt met het feit dat het combineren van *BDDs* aan de basis staat van het uitdrukken van complexe systemen door middel van gecombineerde simpele functies. In Hoofdstuk 3 zal deze techniek uitgelegd worden, het zogenoemde *smelten* (*melding*), dat in Hoofdstuk 4 toegepast zal worden in voorbeelden.

3 Samenvoegen van *BDD*

De term voor het samenvoegen van *BDD* structuren zullen we *smelten* (*melding*) noemen. Het werkt volgens de volgende principes. Men neme twee knopen $\alpha = (v, l, h)$ en $\alpha' = (v', l', h')$. De knoop $\alpha \diamond \alpha'$, het resultaat van de *smelt* actie, de "*emulsie*" (*meld*) van α en α', is dan als volgt gedefinieerd

Figuur 1: fig:samenvoegen



Figuur 2: Voorbeeld: twee *BDDs* die samengesmolten worden door gebruik te maken van formule 1. Let er wel op dat dit voorbeeld geen ontbrekende laag van knopen heeft, waardoor knopen van alle niveaus beschikbaar zijn. Op het moment dat de $\diamond$ ingevuld gaat worden is het ook zaak om de bladeren (zie vergelijking 2) te vereenvoudigen.

als α en α' niet beide bladeren (*sinks*) zijn:

$$\alpha \diamond \alpha' = \begin{cases} (v, l \diamond l', h \diamond h'), & \text{als } v = v'; \\ (v, l \diamond \alpha', h \diamond \alpha'), & \text{als } v < v'; \\ (v', \alpha \diamond l', \alpha \diamond h'), & \text{als } v > v'. \end{cases} \tag{1}$$

Als we deze recursieve formule gebruiken op de wortels van twee *BDDs* f en g, vinden we een diagram dat het paar (f, g) representeert.

De oplettende lezer zal zien dan als je Formule 1 toepast op de bladeren er door samenvoegen van de bladeren —zie bijvoorbeeld voorbeeld in Figuur **??**— in plaats van de twee bladeren $\top$ en $\bot$, er nu vier bladeren mogelijk zijn:

$$\bot \diamond \bot, \quad \bot \diamond \top, \quad \top \diamond \bot, \quad \top \diamond \top \tag{2}$$

Om er weer een "valide" *BDD* van te maken zullen deze bladeren vervangen worden door het uitgerekende blad. Als bijvoorbeeld de $\diamond$ operatie bedoelt is voor een *EN* operatie, wordt de blad-rij in (2) vervangen door de rij $\bot, \bot, \bot, \top$. Nu is het zaak het *BDD* te vereenvoudigen, om zo duplicaatbladeren te snoeien (*pruning*).

De kracht van deze aanpak zit hem in de generieke $\diamond$ operatie. Het maakt niet uit welke booleaanse operatie er gebruikt wordt aan het eind van de rit. De gegeneerde gesmolten *BDD* is geldig voor alle.

Door de *BDDs* achter elkaar te plakken kan met maximaal $B(f)B(g)$ knopen een *BDD* voor $f \diamond g$ gerealiseerd worden. Hoofdstuk 4.2 is hier een voorbeeld van. In het meer algemene geval geldt meestal $B(f) + B(g)$ als bovengrens. Deze grenzen worden in Hoofdstuk 4.1 aangescherpt.

Het smelten ligt aan de basis van de daadwerkelijke synthese. Een simpele impementatie kan gemaakt worden met algoritme R. (a) Maak eerst een reeks van alle knopen α in f en α' in g met knoop $\alpha \diamond \alpha'$ in rij α en column α'. (b) Vervang de bladeren (zie Formule 2) door $\bot$ en $\top$, en (c) voer het algoritme R[1] op $f \diamond g$. Op het eerste gezicht lijkt algoritme R er ongeveer $B(f)B(g)$ operaties over te doen, maar doordat je onbereikbare knopen niet hoeft te evalueren zal je uitkomen op $B(f \diamond g)$.

Deze "truc" zorgt ervoor dat de tijd binnen de perken blijft, maar er is dan nog niets gezegd over de hoeveelheid geheugen die nodig is. Omdat er nu $B(f)B(g)$ knopen in geheugen gehouden moet wordt zal dit problemen opleveren bij kleine en grotere bomen. Om deze efficiëntie aan te pakken is algoritme S[2] ontworpen.

[1] Algoritme R wordt niet in deze samenvatting behandeld; zie de vak-website [SCA2010] waar het algoritme R behandeld wordt door Tom.

[2] Algoritme S wordt niet in deze samenvatting behandeld; het boek [DK2009, p. 90-92] gaat hier echter uitvoerig op in.

4 Voorbeelden

4.1 Product-groei in synthese van *BDD*

Het volgende voorbeeld is een uitwerking van opgave 60 [DK2009, p. 130] de officiële uitwerking is te vinden op [DK2009, p. 195]

Voorgaande aan de uitwerking is het belangrijk te weten wat een *QDD* (aka *quasi-BDD* [DK2009, p. 102-103]) is. Elke functie heeft een unieke *QDD* welke lijkt op een *BDD* echter is de start-knoop altijd ① en elke ⓚ knoop heeft (voor $k < n$) takken naar twee $\boxed{k+1}$ knopen. Wat in het kort wilt zeggen dat elk pad van de start-knoop naar de bladeren een lengte heeft van n. Om dat mogelijk te maken mogen de *HOOG* en *LAAG* takken van een *QDD* gelijk zijn. Let wel op dat reductie toegepast moet worden op tweetallen knopen waarvan de *HOOG* en *LAAG* pointers gelijk zijn.

Neem aan dat $f(x_1, \ldots, x_n)$ en $g(x_1, \ldots, x_n)$ de respectieve *profielen* (*profiles*) [DK2009, p. 101] $(b_0, \ldots, b_n)$ —waar b_k met $0 \leq k < n$ de knopen zijn. Die splitsen op variable $f(x_{k+1})$ en b_n het aantal bladeren is— en $(b'_0, \ldots, b_n)$ hebben, en de respectieve *quasi-profielen* (*quasi-profiles*) [DK2009, p. 103] $(q_0, \ldots, q_n)$ —waar q_{k-1} met $0 < k < n$ het aantal ⓚ knopen in de *QDD* van f is— en $(q'_0, \ldots, q'_n)$.

Om te laten zien dat de gesmolten $f \diamond g$ het aantal knopen

$$B(f \diamond g) \leq \sum_{j=0}^{n} (q_j b'_j + b_j q'_j - b_j b'_j) \tag{3}$$

bevat moet gekeken worden naar het aantal *beads* [DK2009, p. 72] dat mogelijkerwijs gemaakt kan worden van de functies f en g. Voor een willekeurige functie f met als profiel $(b_0, \ldots, b_n)$ en quasi-profiel $(q_0, \ldots, q_n)$ geldt dat $B(b_0, \ldots, b_n) \leq B(q_0, \ldots, q_n)$. En tevens dan $(b_0, \ldots, b_n) \subseteq (q_0, \ldots, q_n)$. Hieruit volgt dat een niet-bead zich bevindt in $(q_0, \ldots, q_n) - (b_0, \ldots, b_n)$.

Elke bead van de orde $n-j$ in het geordende paar (f, g) zal zich bevinden in een van de volgende groepen; (a) de standaard-combinatie van de $b_j b'_j$ geordende beads van (f, g) of (b) de gegenereerde reeks van een (bead,niet-bead) of (niet-bead, bead) $b_j(q'_j - b'_j) + (q_j - bj)b'_j$.

Dit bij elkaar optellen levert op $b_j b'_j + b_j(q'_j - b'_j) + (q_j - bj)b'_j$. Vereenvoudigen en de sommering is oefening voor de lezer.

4.2 Som-groei in synthese van *BDD*

Het volgende voorbeeld is een uitwerking van opgave 63 [DK2009, p. 131] de officiële uitwerking is te vinden op [DK2009, p. 195].

De 2^m-weg multiplexer $M_m(y_1, \ldots, y_m; y_{m+1}, \ldots, y_{m+2^m})$ geeft bij input $y_1, \ldots, y_m$ de waarde van variable y_{m+k} waarbij k het getal is gerepresenteerd door $y_1, \ldots, y_m$, zie [DK2009-Fas0, 7.1.2-(31)]. Hiermee defineren we $\oplus$ de binary operator *XOR* en $n = 2m + 2^m$.

$$\begin{array}{rcl} f(x_1, \ldots, x_n) & = & M_m(x_1 \oplus x_2, x_3 \oplus x_4, \ldots, x_{2m-1} \oplus x_{2m}; x_{2m+1}, \ldots, x_n) \\ g(x_1, \ldots, x_n) & = & M_m(x_2 \oplus x_3, \ldots, x_{2m-2} \oplus x_{2m-1}, x_{2m}; \overline{x}_{2m+1}, \ldots, \overline{x}_n) \end{array} \tag{4}$$

Dan is $B(f) = 2^{m+2} - 1 \approx 4n$, $B(g) = 2^{m+1} - 2^m \approx 3n$ en $B(g \wedge f) = 2^{m+1} + 2^{m-1} - 1 \approx 2n^2$. Om aan deze reeksen te komen is het belangrijk eerst naar de profielen te kijken van f en g om daarna met sommering van deze tot de antwoorden te komen. Als eerste moet opgemerkt worden dat zowel f als g 2^m-weg multiplexers zijn voor welke de profielen zijn $(1, 2, 2, \ldots, 2^{m-1}, 2^{m-1}, 2^m, 1, 1, \ldots, 1, 2)$ respectievelijk
$(0, 1, 2, 2, \ldots, 2^{m-1}, 2^{m-1}, 1, 1, \ldots, 1, 2)$. Dit optellen en afschatten wordt (zie ook [DK2009, p. 82]) $B(f) = 2^{m+2} - 1 \approx 4n$ en $B(g) = 2^{m+1} - 1 \approx 3n$.

De bereking $B(f \wedge g)$ (de werking van de multiplexer) wordt aanzienlijk moeilijker en zal daarom aangestipt worden. We constateren eerst dat er een unieke oplossing bestaat —de oplossing kan aan een uniek getal gekoppeld worden namelijk de invoer— in de vorm van $x_1 \ldots x_{2m}$ als je $((x_1 \oplus x_2)(x_3 \oplus x_4) \ldots (x_{2m-1} \oplus x_{2m}))_2 = p, ((x_2 \oplus x_3) \ldots (x_{2m-2} \oplus x_{2m-1})x_{2m})_2 = q$, waar $0 \leq p, q \leq 2^m$ en $p = q$ dan en slechts als $x_1 = x_3 = \cdots = x_{2m-1} = 0$. Dit zorgt dat het eerste deel —het stuk voor de punt komma— van het profiel van $f \wedge g$ geschreven kan worden als $(1, 2, 4, \ldots, 2^{2m-2}, 2^{2m-1} - 2^{m-1})$. Het tweede deel is aanzienlijk moeilijker, het bestaat uit de sub-functies $x_{2m+j} \wedge \overline{x}_{2m+k}$ of $\overline{x}_{2m+j} \wedge x_{2m+k}$ voor $1 \leq j \leq k \leq 2^m$ tezamen met de originelen x_{2m+j} en $\overline{x}_{2m+j}$ voor $2 \leq j \leq 2^m$. Dat levert het profiel $(2^{m+1} - 2, 2^{m+1} - 2, 2^{m+1} - 4, 2^{m+1} - 6, \ldots, 4, 2, 2)$ op. Beide profielen bij elkaar optellen levert op $B(f \wedge g) = 2^{2m+1} + 2^{m-1} - 1 \approx 2n^2$.

Referenties

[DK2009] D.E. Knuth. Bitwise Tricks & Techniques; Binary Decision Diagrams, volume 4, Fascicle 1, of The Art of Computer Programming. Pearson Education, first edition, 2009.

[DK2009-Fas0] D.E. Knuth. Bitwise Tricks & Techniques; Binary Decision Diagrams, volume 4, Fascicle 0, of The Art of Computer Programming. Pearson Education, first edition, 2009.

[SCA2010] Lecture Seminar Combinatorial Algorithms, `http://www.liacs.nl/~kosters/semcom/`, W.A. (Walter) Kosters, LIACS, Spring 2010.