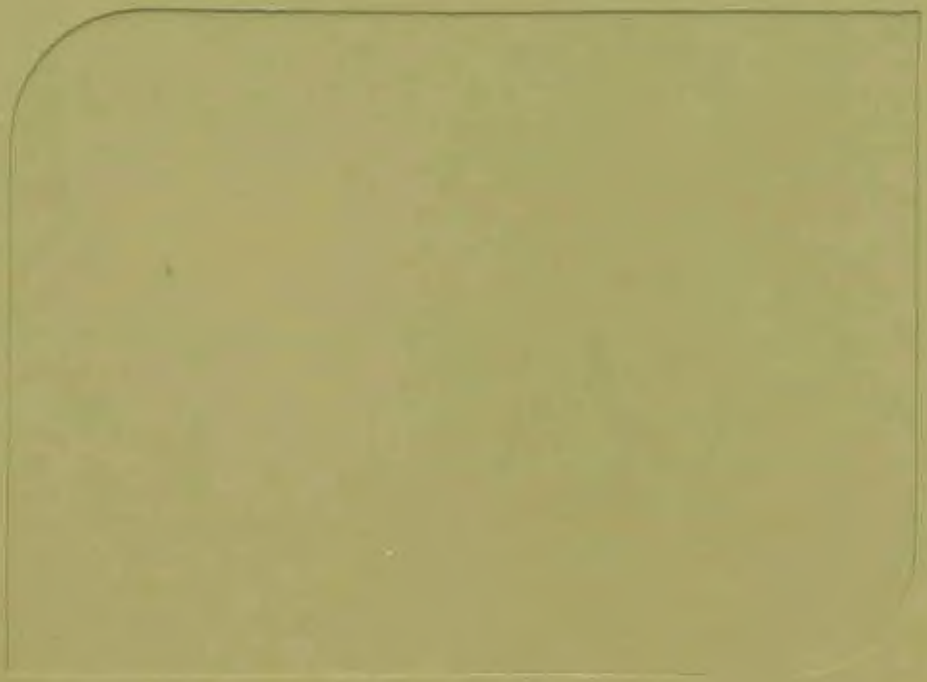


BIM-PROLOG

Joint Project between
BIM
and
Department of Computer Science
Katholieke Universiteit LEUVEN

Sponsored by DPWB/SPPS
under grant nr KBAR/SOFT/1

WIC : Warren Improved Code

by

Gerda JANSSENS **

Internal Report
BIM-prolog IR1

June 1984

* BIM
Kwikstraat 4
B-3078 Everberg Belgium
tel. +32 2 759 59 25

** Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A
B-3030 Heverlee Belgium
tel. +32 16 20 06 56

DPWB = Diensten van de eerste minister : Programmatie van het Wetenschapsbeleid.
SPPS = Services du premier ministre : Programmation de la Politique Scientifique.

WIC
Warren Improved Code

Gerda JANSSENS

Katholieke Universiteit LEUVEN
Departement Computerwetenschappen
Celestijnenlaan 200A
B-3030 HEVERLEE

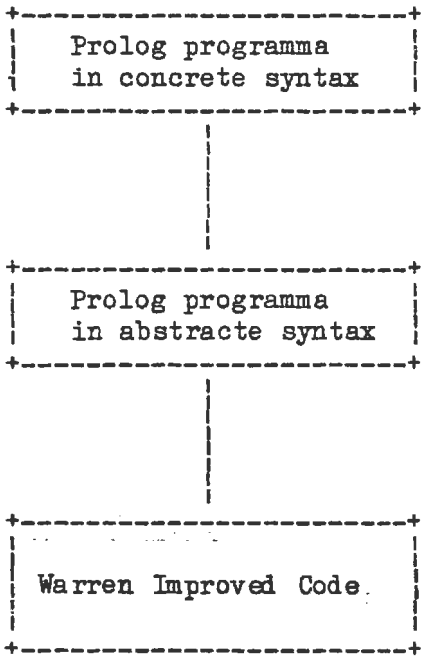
REPORT
BIM-prolog IR1
Juni 1984

1. Inleiding

Wij geven als inleiding een verklaring van de gekozen naam.

Code

Het oorspronkelijke Prolog programma wordt vertaald naar de WIC volgens het volgende schema :



De WIC is uitvoerbaar door de WIC-machine, die in wat volgt beschreven zal worden.

Warren

De WIC is geïnspireerd op de code voorgesteld door Warren[1]. De implementatie ervan heeft de volgende kenmerken :

1) De kopiemethode wordt gebruikt om samengestelde termen voor te stellen die tijdens de uitvoering gebruikt of gekreëerd worden.

2) Op de (locale) stapel staan twee soorten frames : keuzepunten die overeenkomen met de backtrackpunten en omgevings-frames die overeenkomen met de omgevingen waarin plaats voorzien is voor de bindingen van de variabelen.

3) De argumenten van een oproep worden vooraleer de oproep uitgevoerd wordt in argumentenregisters geladen. Bij falen van een unifikatie staan de argumenten van de oproep nog in de argumentenregisters en kunnen zij bij de volgende unifikatie opnieuw gebruikt worden. Bij backtracking moeten de argumentenregisters terug geladen worden : zij worden in de keuzepunten bewaard.

4) Staartrecursie als geheugenoptimisatie is voorzien, en is zelfs veralgemeend tot wat Warren "trimming" van de omgevingen noemt. Dit komt erop neer dat tijdens de uitvoering variabelen in een omgeving in principe vrijgegeven kunnen worden wanneer zij in de nog volgende oproepen niet meer gebruikt worden. Voorwaarde is dat de betreffende omgeving de topomgeving is of dat er geen meer recente keuzepunten op de stapel staan.

5) In de gecompileerde code is er een index op het eerste argument van de Prolog procedure voorzien.

6) Voor de lijststructuur wordt er een speciaal type voorzien zodat de functor niet expliciet moet bijgehouden worden : hij zit reeds vervat in het speciale lijst-type.

Improved

Wij hebben de code voorgesteld door Warren aangepast aan onze ideëen in verband met het professioneel Prolog systeem. Verdere aanpassingen zullen in de toekomst onvermijdelijk zijn : zoals de codegeneratie voor de disjunctie van procedureoproepen in Prolog.

1) Wij voorzien in ons Prologsysteem twee soorten code : gecompileerde (COMP) en te interpreteren. (INTPR). In beide gevallen wordt het Prologprogramma vertaalt naar WIC-instructies.
In de te interpreteren code zal elke oproep van een procedure de functorentabel raadplegen waarin het beginadres van de eerste proceduredefinitie staat. Tijdens de uitvoering is het mogelijk om proceduredefinities weg te laten of toe te voegen.
In de gecompileerde code wordt overeenkomstig met een procedureoproep waarvan het beginadres bekend is bij compilatie, een WIC-instructie gegenereerd die rechtstreeks verwijst naar dat beginadres. Tijdens de uitvoering blijven de proceduredefinities en hun aantal ongewijzigd.

2) Wij willen een interaktie mogelijk maken tussen de te interpreteren en de gecompileerde code.
Wij zullen ook over modules beschikken en het moet dus mogelijk zijn om procedures van een andere module op te roepen.
Het is mogelijk aan te geven dat er voor een bepaalde procedure geen proceduredefinities voorzien zijn en dat een oproep ervan overeenkomt met een falen (UNSPEC).
Een oproep van een onbekende procedure (UNKN) is niet toegelaten en de uitvoering zal stop gezet worden.
Hiervoor voorzien wij in de WIC verschillende soorten procedureoproepen.

- Het suffix "c" duidt op een oproep van een procedure die tijdens het compileren gekend is en die tot dezelfde module behoort.

- Het suffix "u" duidt op een oproep van een procedure die tijdens de compilatie niet gekend binnen de module zelf. Het kan dus een procedure zijn die in een andere te compileren module staat (COMP) of in een te interpreteren module (INTPR) of een procedure die niet nader gespecifieerd is (UNSPEC) of een procedure die effectief een onbekende is.

- Het suffix "e" duidt op de oproep van een ingebouwd evalueerbaar predikaat of van een directief.

3) Wij definiëren andere klassen van variabelen : naast tijdelijk en permanente variabelen herkennen wij ook loze variabelen.

- Een loze variabele is een variabele die slechts één maal voorkomt.

- Een tijdelijke variabele die slechts in één oproep voorkomt. In verband met dit laatste aspect aanzien wij de hoofding en de eerste oproep als één oproep. Het is van geen belang of haar eerste voorkomen in de hoofding, in een structuur of als argument in een oproep staat.
De algemene notatie voor een tijdelijke variabele is Xi.

- Een permanente variabele is een variabele die noch een loze noch een tijdelijke variabele is.
De algemene notatie voor een permanente variabele is Yi.

4) De types die toegekend kunnen worden aan Prolog termen zijn aangepast en zullen in wat volgt beschreven worden.

2. Beschrijving van de WIC-machine

In wat volgt zullen die runtime-structuren, de machineregisters en de voorstellingswijze van de Prolog termen beschreven worden.

2.1. Runtime-structuren van de WIC-machine

Voor de uitvoering van de WIC-instructies die voor een Prolog programma zijn gegenereerd, gebruikt de WIC-machine de traditionele runtime-structuren zoals de locale stapel, de kopiestapel, de herstartstapel en de codetabel.

2.1.1. Herstartstapel

De herstartstapel bevat de bindingen die bij backtracking terug vrijgemaakt moeten worden.
Het register TR (trail) duidt de top van de herstartstapel aan.

2.1.2. Kopiestapel

De kopiestapel bevat de samengestelde termen die tijdens de uitvoering in de argumentenregisters geladen worden of die tijdens de uitvoering gekreëerd worden.
Het register H (heap) duidt de top van de kopiestapel aan.
Het register HB (heap-back) duidt de top van de kopiestapel aan toen het meest recente keuzepunt op de stapel werd gezet.

2.1.3. Codetabel

In de codetabel houden wij de WIC bij die gegenereerd is voor de gecompileerde en de te interpreteren procedures. De codetabel is zodanig georganiseerd dat er op niveau van de procedures tijdens de uitvoering te compileren code kan toegevoegd worden en dat er ook te interpreteren code kan toegevoegd en weggelaten worden. Merk op dat eveneens proceduredefinities kunnen weggelaten of toegevoegd worden aan de te interpreteren procedures.
Het register P (programmateller) duidt de WIC-instructie aan die uitgevoerd wordt.
Het register CP (continuation wijzer) duidt de WIC-instructie aan die uitgevoerd moet worden als de huidige oproep is afgewerkt.

2.1.4. Locale stapel

De locale stapel zullen wij in wat volgt ook kortweg de stapel noemen. Het register E (environment) duidt de top van de locale stapel aan.
Op de stapel staan zoals reeds vermeld werd twee soorten knooppunten :

- Een keuzepunt komt overeen met een backtrackpunt.
  In een keuzepunt houden wij de volgende informatie bij :
  - len : de lengte van het keuzepunt die afhankelijk is van het aantal argumenten van de oproep.
  - alt : de eerstvolgende alternatieve clause.
  - back : vorig keuzepunt.

- copy : top van de kopiestapel op het moment van de kreatie van dit keuzepunt.

- reset : top van de herstartstapel op het moment van de kreatie van dit keuzepunt.

- cenv : waarde van E op het moment van de kreatie van dit keuzepunt.

- ccont : waarde van CP op het moment van de kreatie van dit keuzepunt.

- de argumentenregisters die geladen werden voor de uitvoering van de oproep waarvoor dit backtrackpunt gekreëerd werd.

- Een omgevings-frame wordt op de stapel gezet indien de geselecteerde proceduredefinitie meer dan één oproep in het lichaam heeft. Het omgevings-frame bevat de volgende velden :

  - eenv : verwijzing naar het vorige omgevings_frame

  - econt : waarde van CP op het moment van de kreatie van dit omgevings-frame.

  - plaats voor de voorstelling van de bindingen van de permanente variabelen.

Het register B (back) duidt het meest recente keuzepunt aan dat op de stapel staat.
Het register E duidt het meest recente omgevings-frame aan dat op de stapel staat.
Het register A duidt de top van de stapel aan.

2.2. Registers van de WIC-machine

Naast de reeds vermelde registers (TR, H, HB, P, CP, B, E, A) gebruikt de WIC-machine nog de volgende registers :

- S : wordt gebruikt bij de unifikatie tussen twee samengestelde termen. De samengestelde term die overeenkomt met het argument van de oproep wordt overlopen met behulp van het register S dat de achtereenvolgende argumenten van de samengestelde term aanduidt.

- A1,...,Ai,...,An : de argumentenregisters waarin de argumenten van de oproep voor de uitvoering van die oproep geladen worden.

- X1,...,Xj,...Xm : de registers die gebruikt worden voor de voorstelling van de binding van de tijdelijke variabelen.

Wij kunnen de WIC-code zodanig optimaliseren dat er in feite maar één stel registers nodig is.

- CFL (cutflag) : wordt gebruikt bij de uitvoering van een cut-operatie.

2.3. Voorstelling van de Prolog termen.

De Prolog termen worden gekenmerkt door een type-veld en een waarde-veld. Wij geven een opsomming van de mogelijke types en hun overeenkomstige waarde-velden.

```
voorstelling van een geheel getal :
     INT            gehele getal (< 2 tot de macht 24)
voorstelling van een reëel getal :
     REAL           reëel getal
voorstelling van een constante of een atoom
     CONST          referentie naar de constantentabel
voorstelling van een lijst :
     LIST           referentie naar de argumentenlijst
                    op de kopiestapel
voorstelling van een samengestelde term :
     STRUCT         referentie naar de functor gevolgd door de
                    argumentenlijst op de kopiestapel
voorstelling van een ongedefinieerde waarde :
     UNDEF
voorstelling van een interne referentie :
     SREF           verwijzing naar de stapel
     KREF           verwijzing naar kopiestapel
```

3. WIC-instructies

In deze paragraaf geven wij eerst een overzicht van de soorten WIC-instructies, vervolgens bespreken wij elke klasse van WIC-instructies apart. In de volgende paragrafen wordt de pseudo-code overeenkomstig met elke WIC-instructie vermeld en een aantal niet triviale instructies worden uitvoeriger besproken.

3.1. Overzicht van de soorten instructies

Voor elke klasse van WIC-instrcties geven wij de benaming, de elementen van een Prolog procedure waarmee zij overeenkomen en wat hun uitvoering moet bewerkstelligen.

1) De get-instructies komen overeen met de argumenten van de hoofding van een proceduredefinitie. Zij zorgen voor de unifikatie tussen de argumenten van de hoofding en de overeenkomstige argumentenregisters.

2) De put-instructie komen overeen met de argumenten van een procedureoproep. Zij laden de actuele waarde van de argumenten in

de argumentenregisters.

3) De unify-instructies komen overeen met de argumenten van een samengestelde term. Hun uitvoering zorgt ofwel voor de unifikatie van een samengestelde term beschreven op de kopiestapel ofwel voor de constructie van de samengestelde term op de kopiestapel.

4) De procedurale instructies komen overeen met de predikaten in de hoofding of in het lichaam van een proceduredefinitie. Hun uitvoering heeft betrekking op transfer van de controle, staat in voor de allocatie en deallocatie van omgevings-frames en ook voor de eigenlijke procedureoproepen.

5) De index- instructies stemmen overeen met de Prolog procedures. Zij groeperen de proceduredefinities van eenzelfde procedure en zorgen voor de indexing op het eerste argument van de gecompileerde proceduredefinities.

3.2. Get-instructies

Wij sommen de get-instructies op en geven tegelijkertijd aan voor welk soort argument zij gebruikt worden.

get-variable Yn,Ai	eerste voorkomen van de permanente variabele Yn is het i-de argument.
get-value Yn,Ai	een volgend voorkomen van Yn als i-de argument
get-variable Xn,Ai	eerste voorkomen van de tijdelijke variabele Xn als i-de variabele
get-value Xn,Ai	een volgende voorkomen van Xn als i-de argument
get-int I,Ai	het geheel getal I als i-de argument
get-constant C,Ai	de constante C als i-de argument
get-list Ai	het i-de argument is een lijst
get-struct F,Ai	het i-de argument is een samengestelde term met functor F

De suffixen variable, value, int, constant, list en struct hebben bij de volgende klassen dezelfde betekenis.
Merk op dat er voor een loze variabele als argument van een hoofding geen overeenkomstige WIC-instructie gegenereerd wordt. Er moet voor dat argument geen unifikatie gebeuren.

3.3. Put-instructies

Overzicht van de put-instructies :

put-variable Yn,Ai	put-variable Xn,Ai
put-value Yn,Ai	put-value Xn,Ai
put-unsafe-value Yn,Ai	put-void Ai
put-int I,Ai	put-constant C,Ai
put-list Ai	put-struct F,Ai

De instructie "put-void Ai" komt overeen met een loze variabele als i-de argument van de oproep.
In de laatste oproep waarin een permanente variabele voorkomt, wordt de eerste "put-value Yn,Ai" vervangen door een "put-unsafe-value Yn,Ai" (Wij noemen dit een "onveilige" variabele). Tijdens de uitvoering gaat men bij deze instructie na of er een verwijzing naar de eigen omgeving in het argumentenregister zou geladen worden. Dit moet vermeden worden vermits de trimming een deel van de omgeving vóór de eigenlijke uitvoering van de oproep zal weglaten indien het het meest recente omgevings-frame op de stapel betreft. Wij zullen in dat geval de onveilige variabele op de kopiestapel zetten.

3.4. Unify-instructies

Overzicht van de unify-instructies :

```
unify-variable Yn,Ai          unify-variable Xn,Ai
unify-value Yn,Ai             unify-value Xn,Ai
                              unify-void Ai
unify-int I,Ai                unify-constant C,Ai
unify-list Ai                 unify-struct F,Ai
```

De unify-instructies worden gebruikt om de argumenten van een samengestelde term te specifiëren. De samengestelde term is ofwel een lijststructuur ofwel een structuur met een willekeurige functor F.
Deze instructies worden voorafgegaan door een instructie die de samengestelde term kenmerkt : get-list, get-struct, put-list, put-struct, unify-list of unify-struct. Deze instructie bepaalt de mode waarin de unify-instructies moeten geïnterpreteerd worden : READ-mode of WRITE-mode.
Indien de waarde van het gespecifieerde argumentenregister een lijst of een samengestelde term is, wordt de READ-mode geaktiveerd. Indien het argumentenregister verwijst naar een ongedefinieerde variabele, wordt de WRITE-mode geaktiveerd.
In de READ-mode gebeurt er unifikatie tussen de argumenten beschreven door de unify-instructies en de argumenten van een samengestelde term op de kopiestapel waarvan de argumenten aangeduid worden door het S-register.
In de WRITE-mode geven de unify-instructies een beschrijving van een samengestelde term die op de kopiestapel moet gezet worden waarbij de variabelen eventueel worden vervangen door hun waarde op dat moment van de uitvoering.

Wij geven een voorbeeld van de code die genereerd wordt voor een geneste samengestelde term :

het i-de argument van een oproep is "F(.(7,.(X,nil)),3,F(X,Y,Z))". de nummers overeenkomstig met X,Y en Z zijn 1,2 en 3.

```
put-struct F,Ai
unify-variable Xf1
unify-int 3
unify-variable Xf2
unify-list Xf1
unify-int 7
unify-variable Xf3
unify-list Xf3
unify-value Y1
unify-constant nil
unify-struct F,Xf2
unify-value Y1
unify-value Y2
unify-value Y3
```

De tijdelijke variabelen Xf1, Xf2 en Xf3 komen niet overeen met een variabele in het oorspronkelijk Prolog-programma, zij worden enkel gebruikt bij de codegeneratie voor geneste samengestelde termen.

3.5. Procedurale instructies

Overzicht van de procedurale instructies :

proceed	allocate	
execute P	deallocate	dealex P
call P,n	lastcut	

De allocate-instructie doet de allocatie van een omgevings-frame. Zij is de eerste instructie voor een proceduredefinitie met meer dan één oproep in haar lichaam.
De huidige waarde van E , CP en CFL worden bijgehouden. Indien er permanente variabelen in de proceduredefinitie voorkomen, komen die op de stapel in het nieuwe omgevings-frame.

De deallocate-instructie zal de oude waarden van E en CP herstellen. Het omgevings-frame is overbodig en zal vrijgegeven worden indien het op de top van de stapel staat.
Een veel voorkomend gebruik van deallocate is dat voor de laatste oproep de vaderomgeving kan vrijgegeven worden als het de topomgeving is. (Dit is in feite de implementatie van staartrecursie). De sequentie deallocate en execute P wordt één instructie : dealex P.
Een minder triviaal gebruik van allocate en deallocate is dat in het geval van een cut-operatie als (laatste) oproep. Dan wil men de vaderomgeving op de stapel staat terwijl de (laatste) oproep wordt uitgevoerd. (cfr. uitleg over de cut-operatie)

De proceed-instructie komt overeen met een ledig lichaam van een proceduredefinitie.

De "execute P"-instructie komt overeen met de laatste oproep in het lichaam. P geeft aan welke procedure opgeroepen wordt.

De "call P,n"-instructie komt overeen met een oproep van P die niet de laatste is in het lichaam. In de nog volgende oproepen komen nog n permanente variabelen voor die de nummers 1 tot en met n hebben. Van de permanente variabelen in het omgevings-frame zullen alleen de eerste n nog gebruikt worden : de rest kan vrijgegeven worden indien het de topomgeving is.
Op deze manier gebeurt de " trimming " : de grootte van het omgevings-frame is dynamisch vermits bij elke oproep eventueel een deel kan vrijgegeven worden dankzij een aangepaste nummering van de permanente variabelen.

De suffixen -c, -u en -e worden aan call, execute en dealex toegevoegd.

Als de laatste oproep in het lichaam een cut is dan wordt de lastcut- instructie gebruikt.

Als de laatste oproep in het lichaam een metacall is, dan mag er geen dealexe gebruikt worden om een eventuele cut-operatie nog correct uit te voeren : er wordt dan een calle gevolgd door een deallocate gegenereerd.

3.6. Index-instructies

Overzicht van de index-instructies :

try-me-else C,i	try C,i
retry-me-else C	retry C
trust-me-else	trust C
try-me-only C	

switch-on-term Cr,Cc,Cl,Cs

ortry C,n orretry C jump C

Deze instructies zorgen onder andere voor de groepering van de proceduredefinities per procedure. Alle proceduredefinities worden aan elkaar geketend zoals in het volgende voorbeeld.

```
C0:  try-me-else C1        kreëer een keuzepunt, C1 is adres van de
     .                     eerstvolgende alternatieve proceduredefinitie
     WIC-code voor proceduredefinitie0
     .
C1:  retry-me-else C2      pas alternatief aan in keuzepunt : C2
     .
     WIC-code voor proceduredefinitie1
     .
C2:  trust-me-else         weglaten van keuzepunt indien op top
     .                     van de stapel
     WIC-code voor proceduredefinitie2
     .
```

Daarop wordt een indexing op het eerste argument van de hoofding gesuperponeerd in het geval van de gecompileerde code.

switch-on-term Cr,Cc,Cl,Cs

- Cc : adres van de proceduredefinities waarvan het eerste argument een constante is.
- Cl : adres van de proceduredefinities waarvan het eerste argument een lijst is.
- Cs : adres van de proceduredefinities waarvan het eerste argument een structuur is.
- Cr : adres van de eerste proceduredefinitie : als het eerste argument een variabele is, kan er geen selectie gebeuren.

Indien er meer dan één definitie is met bijvoorbeeld een constante als eerste argument, dan is Cc het adres van een reeks try, retry en trust instructies. Als er juist één proceduredefinitie is, is Cc het adres van die definitie. Deze opmerking geldt ook voor Cl en Cs.

In het geval dat er juist één proceduredefinitie is voor een procedure, moet er in de te interpreteren versie een keten van één element opgebouwd worden door de try-me-only instructie. Elke proceduredefinitie wordt in de te interpreteren versie als het ware door een hoofding voorafgegaan. Bij latere toevoeging van definities wordt deze hoofding aangepast. In de gecompileerde

versie liggen de definities vast en is een try-me-only instructie overbodig : er is geen hoofding nodig die later aangepast kan worden.

Een of-lijst wordt vertaald met behulp van de ortry- , prretry- , trust-me-else- en de jump- instructies.
Voor ...(a;b,c;d),e,.. wordt de volgende code gegenereerd :

```
C0 : ortry C1,0        kreëer een keuzepunt (zonder arg.reg.)
                       C1 is adres van begin van de volgende
                       tak in de of-lijst.
     callc a,0
     jump C3           spring naar de oproep na de of-lijst
C1 : orretry C2        pas alternatief aan in keuzepunt
     callc b,0
     callc c,0
     jump C3
C2 : trust-me-else     weglaten van keuzepunt indien op
                       top van de stapel.
     callc d,0
C3 : callc e,0
     ..
```

De init- instructie wordt gegenereerd voor al de permanente variabelen die voor de eerste maal voorkomen in een tak van de of-lijst en die niet in alle takken voorkomen en die ook nog na de of-lijst voorkomen.

4. Pseudocode voor de WIC-instructies

Wij geven in deze paragraaf de pseudo-code voor de WIC-instructies. In een volgende paragraaf zullen wij de volgende onderwerpen meer in detail bespreken : loze variabelen, de cut-operatie en de pseudo-code van proceed.

callc procedure,n

```
if ( E<B) then A <- E+n+2;          /* E<B d.w.z. E recenter dan B */
else A <- B + lengte-keuzepunt;
CP <- next-instructie;              /* CP verwijst eventueel naar een
                                       reeks put-instructies gevolgd door een
                                       call-instruktie of een execute-instructie */
P <- procedure;
set-cutflag off
```

callu ftentrynr,n

```
if ( E<B) then A <- E+n+2;          /* E<B d.w.z. E recenter dan B */
else A <- B + lengte-keuzepunt;
CP <- next-instructie;            /* CP verwijst eventueel naar een
                        reeks put-instructies gevolgd door  een
                        pcall-instruktie of een execute-instructie */
P <- zoek-procedure-in-functorentabel(ftentrynr);
set-cutflag off
```

calle nrevalpred,n

```
P <- next-instruktie;
evalpred(nrevalpred,n);
          /* indien nodig (vb. bij een niet deterministisch
           predikaat, of bij een consult die aanleiding geeft
           tot een nieuwe query */
          zal A eerst aangepast worden */
set-cutflag off
```

try-me-else C,I

```
/* create een backtrackpunt van af A */
/* zet op de stack : */
   lengte-van-keuzepunt <- cte + I
   alternatieve clause  <- C
    H,TR,B,CP,E,de registers Ai
B <- A;
HB <- H;
A <- B + lengte-keuzepunt;
P <- P + 1;
set-cutflag on
```

retry-me-else C

```
alternatief aanpassen : C
A <- B+lengte-keuzepunt;
P <- P+1;
set-cutflag on
```

trust-me-else

```
A <- B;
reset(B);update(HB);
P <- P+1;
```

ortry C,n

```
if (E<B) then A <- E+n+2;
else A <- B + lengte-keuzepunt;
/* kreëer een keuzepunt vanaf A */
 /* zet op de stapel */
    lengte-van-keuzepunt
    alternatief <- C
    H,TR,B,CP,E
B <- A;
HB <- H;
A <- B + lengte-keuzepunt;
P <- P +1;
```

orretry C

```
alternatief aanpassen <-  C;
A <- B + lengte-keuzepunt;
P <- P+1;
```

try C,I

```
/* create backtrackpoint vanaf A */
/* zet op de stack : */
     lengte-keuzepunt <- cte + I
     alternatieve clause <- next-instructie
     H,TR,B,CP,E,registers Ai
B <- A;
HB <- H;
A <- B + lengte-keuzepunt;
P <- C;
set-cutflag on
```

retry C

```
alternatief aanpassen : next-instructie
A <- B + lengte-keuzepunt;
P <-C;
set-cutflag on
```

trust C

```
A <- B;
reset(B);update(HB);
P <- C;
```

switch-on-term(Cr,Cc,Cl,Cs)

```
regderef(adres,A(1));
switch(ageeftype(adres)) {
case INT :
case CONST :
     if ( Cc == 0) falen();
     else P <- P+Cc;
     break;
case LIST :
     if (Cl == 0) falen();
     else P <- P+Cl;
     break;
case STRUCT :
     if (Cs == 0) falen();
     else P <- P+Cs;
     break;
default : P <- P+Cr; break;
};
```

allocate

```
vanaf A wegbergen van E en CP
   en wegbergen van cut-flag
E <- A;
P <- P +1;
```

deallocate

```
if (E<B) then A <- E;
else A <- B + lengte-keuzepunt;
reset E en CP vanuit E;
P <-CP;
set cutflag off;
```

proceed

```
P <- CP
```

jump

```
P <- C;
```

lastcut

```
while (B<E) reset(B);
if ( cutflag in E on) reset(B);
A <-E;
reset E en CP vanuit E;
set-cutflag off;
P <- CP;
```

init n

```
stack tkentoe(E+n,UNDEF);
P <- P+1;
```

```
executec C

P <- C;
set-cutflag off

executeu ftentrynr

P <- zoek-procedure-in-functorentabel(ftentrynr);
set-cutflag off

executee nrevalpred

P <- CP;
evalpred(nrevalpred,-3);
set-cutflag off

dealexc C

if ( E<B) then A <- E;
else A <- B + lengte-keuzepunt;
reset E en CP vanuit E;
P <- C;
set-cutflag off

dealexu ftentrynr

if (E<B) then A <- E;
else A <- B + lengte-keuzepunt;
reset E en CP vanuit E;
P <- zoek-procedure-in-functorentabel(ftentrynr);
set-cutflag off

dealexe nrevalpred

reset E en CP vanuit E
P <- CP;
evalpred(nrevalpred,-2);
set-cutflag off

backtrack

/* opgeroepen bij falen van de unifikatie en bij oproep
van fail */
if gedefinieerd(B) {
      restore P(<- alternatieve clause )
               H,TR,CP,E,registers Ai
     reset variabelen
    }
else gedaan = TRUE
set-cutflag off
```

get-variable Yn, Ai

```
regderef(adres,Ai);
if ( ageeftype(adres) == UNDEF) stack-kentoe(E+n,ref,adres);
else stack-move(E+n,adres);
```

get-variable Xn, Ai

```
regderef(adres,Ai);
if ( ageeftype(adres) == UNDEF) reg-kentoe(Xn,ref,adres);
else reg-move(Xn,adres);
```

get-value Yn,Ai

```
yderef(uadres,E+n);
regderef(adres,Ai);
unify(uadres,adres);
```

get-value Xn,Ai

```
regderef(uadres,Xn);
regderef(adres,Ai);
unify(uadres,adres);
```

get-constant C,Ai

```
regderef(adres,Ai);
switch ( ageeftype(adres)){
case  UNDEF : apr-kentoe(adres,CONST,C); break;
case CONST : if( !( ageefwaarde(adres) == C) falen-unif();
             break;
default : falen_unif();
}
```

get-int I,Ai

cfr. get-constant C,Ai

get-list Ai

```
regderef(adres,Ai);
switch( ageeftype(adres) ){
case UNDEF : apr-kentoe(adres,LIST,H); mode = WRITE; break;
case LIST : mode = READ; S = linit(adres); break;
default : falen-unif(); break;
}
```

get-structure F,Ai

```
regderef(adres,Ai)
switch( ageeftype(adres)) {
case UNDEF : apr-kentoe(adres,STRUCT,H); kopie-push(fn,F);
             mode = WRITE;break;
case STRUCT : if( kopie-geefwaarde( ageefwaarde(adres)) == F)
     { mode = READ; S = sinit(adres); }
     else falen-unif();
     break;
default : falen-unif(); break;
}
```

put-variable Yn,Ai

```
reg-kentoe(Ai,sref,E+n);
stack-undef(E+n);
  /* adres E+n op de stack wordt op UNDEF gezet */
```

put-variable Xn, Ai

```
reg-kentoe(Ai,kref,H); reg-kentoe(Xn,kref,H);
kopie-undef();
```

put-value Yn,Ai

```
yderef(adres,E+n);
if ( ifundefop( adres) ) reg-kentoe(Ai,ref,adres);
else reg-move(Ai,adres);
```

put-value Xn,Ai

```
regderef(adres,Xn);
if ( Ifundefop(adres)) reg-kentoe(Ai,ref,adres);
else reg-move(Ai,adres);
```

put-unsafe-value Yn,Ai

```
yderef(adres,E+n);
if ( unsafe(adres))
     /* dit wil zeggen adres is een adres-op-stack,
        recenter dan E
        en het typeveld op adres heeft als waarde UNDEF */
  {
     stack-kentoe(adres,kref,H);
     reg-kentoe(Ai,kref,H);
     kopie-undef();
     }
else if (ageeftype(adres) == UNDEF ) reg-kentoe(Ai, ref, adres);
else reg-move(Ai,adres);
```

put-constant C,Ai

```
reg-kentoe(Ai,CONST,C)
```

put-int I,Ai

```
reg-kentoe(Ai,INT,I)
```

put-void Ai

```
reg-kentoe(Ai,kref,H);
kopie-undef();
```

put-list Ai

```
reg-kentoe(Ai,LIST,H); mode = WRITE
```

put-struct F,Ai

```
reg-kentoe(Ai,STRUCT,H);
kopie-push(fn,F); mode = WRITE;
```

unify-variable Yn

```
switch(mode) {
case READ :
     sderef(adres,S);
     if ( kopie-geeftype(adres) == UNDEF)
        stack-prkentoe(E+n,kref,adres);
     else stack-prkentoe(E+n,kopie-geeftype(adres),
                         kopie-geefwaarde(adres));
     kopie-update(S); break;
case WRITE :
     stack-prkentoe(E+n,kref,H);
     kopie-undef(); break;
}
```

unify-variable Xn

```
switch(mode) {
case READ :
     sderef(adres,S);
     if(kopie-geeftype(adres) == UNDEF)
        reg-kentoe(Xn,kref,adres);
     else reg-kentoe(Xn,kopie-geeftype(adres),
                     kopie-geefwaarde(adres));
     kopie-update(S);break;
case WRITE :
     reg-kentoe(Xn,kref,H);
     kopie-undef();break;
     }
```

unify-value Yn

```
switch(mode) {
case READ :
     yderef(uadres,E+n); sderef(adres,S);
     unify(uadres,adres);
     kopie-update(S);break;
case WRITE :
     yderef(adres,E+n);
     if( ifundefop(adres) )
       { apr-kentoe(adres,kref,H); kopie-undef();}
     else kopie-move(adres);
     break;
}
```

unify-value Xn

```
switch(mode) {
case READ :
     regderef(uadres,Xn); sderef(adres,S);
     unify(uadres,adres); kopie-update(S);break;
case WRITE :
     regderef(adres,Xn);
     if( ifundefop(adres))
       {apr-kentoe(adres,kref,H); kopie-undef();}
     else kopie-move(adres); break;
}
```

unify-void

```
switch (mode) {
case READ : kopie-update(); break;
case WRITE : kopie-undef(); break;
}
```

unify-constant C

```
switch(mode) {
case READ :
     sderef(adres,S);
     switch(kopie-geeftype(adres) ){
     case UNDEF : kopie-kentoe(adres,CONST,C);break
     case CONST : if(! kopie-geefwaarde(adres) == C) falen-unif();
                  else kopie-update(S);
                  break
     default : falen-unif();
               break;
     }
case WRITE kopie-push(CONST,C); break;
}
```

unify-int I

cfr. unify-constant C

unify-list Xn

```
/* regderef(adres,Xn) is onnodig : Xn is een gereserveerde var.*/
switch( ageeftype(adres)) {
case KREF : apr-kentoe(ageefwaarde(adres),LIST,H);
            mode = WRITE;break;
case LIST : mode = READ; S = linit(adres); break;
default : falen-unif(); break;
}
```

unify-struct F,Xn

```
switch( ageeftype(Xn)) {
case KREF : apr-kentoe(reg-geefwaarde(Xn),STRUCT,H);
            kopie-push(fn,F); mode = WRITE; break;
case STRUCT :if (kopie-geefwaarde(reg-geefwaarde(Xn)) == F)
                { mode = READ; S = sinit(Xn); break;
default : falen-unif();
     break;
}
```

```
unify(uadres,adres)

/* beide argumenten van unify zijn juist voor de oproep van  unify
gedereferencieerd. */

switch(ageeftype(uadres)) {

case INT :
switch( ageeftype(adres)) {
case UNDEF : apr-move(adres,uadres);break;
case INT : if(!(ageefwaarde(uadres) == ageefwaarde(adres))
             falen-unif(); break;
default :falen-unif(); break;
} ; break;


case CONST :
switch(ageeftype(uadres)) {
case UNDEF : apr-move(adres,uadres);break;
case CONST : if(!(ageefwaarde(uadres) == ageefwaarde(adres))
             falen-unif(); break;
default :falen-unif(); break;
} ; break;


case UNDEF :
switch( ageeftype(adres)) {
case UNDEF :
if ( kopie-adres(uadres))
  if ( kopie-adres(adres) )
    { if ( kopie-morerecent(uadres,adres))
        kopie-prkentoe(uadres,kref,adres);
      else kopie-prkentoe(adres,kref,uadres);
    }
  else /* adres is een adres op stack */
    stack-prkentoe(adres,kref,uadres);
else /* uadres is een adres op stack */
  if ( kopie-adres(adres) )
    stack-prkentoe(uadres,kref,adres);
  else if (stack-morerecent(uadres,adres)
         stack-prkentoe(uadres,sref,adres);
       else stack-prkentoe(adres,sref,uadres);
break;
default : /* case INT,CONST,LIST,STRUCT */
     apr-move(uadres,adres);break;
};break;
```

```
case LIST :
switch( ageeftype(adres)) {
case UNDEF : apr-move(adres,uadres); break;
case LIST :  uadres = ageefwaarde(uadres);
     adres = ageefwaarde(adres);
     for( arity = 2; !falen && arity > = 1; arity--)
     { sderef(huadres,kopie-aftupdate(uadres));
       sderef(hadres,kopie-aftupdate(adres));
       unify(huadres,hadres);
     };break;
default : falen-unif(); break;
};break;


case STRUCT :
switch( ageeftype(adres)) {
case UNDEF : apr-move(adres,uadres);break;
case STRUCT : uadres = ageefwaarde(uadres);
     adres = ageefwaarde(adres);
     if ( (entry = kopie-geefwaarde(uadres)) ==
                          kopie-geefwaarde(adres))
     for( arity =ft-zoekarity(entry); !falen && arity >=1 arity--)
         { sderef( huadres,kopie-befupdate(uadres));
           sderef(hadres,kopie-befupdate(adres));
           unify(huadres,hadres);
         }
     else falen-unif(); break;
default ; falen-unif(); break;
};break;
}

}
```

5. Meer in detail

5.1. Variabelen

Een loze variabele is een variabele die slechts 1 maal voorkomt in de clause.

Warren gebruikt slechts twee klassen om zijn variabelen te klassificeren : tijdelijke en permanente variabelen.
De behandeling van een loze variabele volgens Warren is als volgt:

- Als ze in de hoofding voorkomt in een structuur : unify-variable Xn. Dit houdt in dat het een tijdelijke variabele is en dat de unifikatie zal gebeuren alsof het een tijdelijke variabele betreft, alhoewel er eigenlijk geen unifikatie hoeft plaats te vinden.

- Als ze in de hoofding voorkomt maar niet in een structuur : geen overeenkomstige instructie noodzakelijk. Dit houdt in dat de unifikatie met een loze variabele als argument in de hoofding niet gebeurt.

- Als ze in een structuur in het lichaam voorkomt, of als ze in de laatste oproep als argument voorkomt, dan is het een tijdelijke variabele. De respectievelijke instructies zijn : unify-variable Xn en put-variable Xn,Ai.

- Als ze noch in de hoofding, noch in de laatste oproep, noch in een structuur voorkomt, is het een permanente variabele :

  - Als ze als argument in een oproep voorkomt, zouden wij overeenkomstig met haar eerste voorkomen een put-variable Yn,Ai moeten genereren en overeenkomstig met haar laatste voorkomen een put-unsafe-value Yn,Ai.

  - Als ze in een structuur voorkomt : unify-variable Yn.

Wij stellen voor om in de WIC code de loze variabelen apart te behandelen : naast de tijdelijke en de permanente variabelen herkennen wij nu ook loze variabelen. De loze variabelen worden dan als volgt behandeld :

- Voor een loze variabele die als argument in een hoofding voorkomt, wordt geen get-instructie gegenereerd : er hoeft geen unificatie te gebeuren.

- Voor een loze variabele in een structuur in de hoofding of in het lichaam gegenereren wij een unify-void instructie.
  In de READ-mode hoeft er geen unificatie te gebeuren.
  In de WRITE-mode zetten wij een undef op de kopiestapel.

- Voor een loze variable als argument van een oproep genereren wij een put-void Ai. Vermits een argumentenregister geen undef mag bevatten, zetten wij een undef op de kopiestapel en laten wij Ai ernaar verwijzen.

Merk op dat wij ook de klasse van de tijdelijke variabelen hebben uitgebreid.
De voorwaarde van Warren is dat de variabele haar eerste voorkomen in de hoofding, een structuur of in de laatste oproep heeft en dat zij maar in één oproep voorkomt.
Stel dat het eerste voorkomen van de variabele V in de oproep van q staat en dat V maar in één oproep voorkomt. Deze definitie geeft aanleiding tot de volgende situatie :

```
..,q(V,F(V,s),..),..      <->   ...,q(F(V,s),V,..),..
 V is permanent                    V is tijdelijk
```

Vanuit het standpunt van de codeoptimisatie is deze "asymetrische" klassifikatie onwenselijk.
Onze definitie is dat V een tijdelijke variabele is als zij in één oproep voorkomt zodat V in beide gevallen een tijdelijke variabele is.

5.2. De cut-operatie

5.2.1. Frames op de (locale) stack

Op de locale stack staan twee soorten frames : omgevings-frames en keuzepunten.

Een keuzepunt wordt gekreëerd als de procedure die opgeroepen wordt meer dan één proceduredefinitie heeft : naar aanleiding van een try-me-else of een try-instructie. Het keuzepunt bevat :

- de lengte van het keuzepunt die afhankelijk is van het aantal argumenten van de oproep.
- de eerstvolgende alternatieve clause.
- H, TR, B, CP, E
- argumentenregisters Ai.

Een omgevings-frame wordt gekreëerd als de geselecteerde proceduredefinitie meer dan één oproep in het lichaam heeft : naar aanleiding van de allocate-instructie. Het omgevings-frame bevat de waarde van E en CP en later zullen de permanente variabelen hieraan toegevoegd worden. E wijst naar het nieuwe omgevings-frame.

Tijdens de uitvoering is de chronologische volgorde van de instructies de volgende :

```
call P,n
(try-me-else C) of (try C)        -> keuzepunt
(allocate)                        -> omgevings-frame
```

Bij de uitvoering van een oproep zijn de volgende stackmanipulaties mogelijk :

- een keuzepunt en een omgevings-frame worden op de stapel gezet.
- enkel een keuzepunt wordt op de stapel gezet.
- enkel een omgevings-frame wordt op de stapel gezet.

- geen enkel frame wordt op de stapel gezet.

5.2.2. Definitie van de cut-operatie

Een cut-operatie verwijdert alle alternatieven vanaf zijn vader-oproep, zelfs als de cut-operatie in een disjunctie voorkomt.

5.2.3. Implementatie van de cut-operatie

Om een cut-operatie uit te voeren is er informatie nodig die aangeeft of een omgevings-frame een bijhorend keuzepunt heeft. (of een oproep zowel een keuzepunt als een omgevings-frame op de stack heeft gezet)

Wij gebruiken een globale, booleaanse variabele CFL : "cutflag".

Als cutflag de waarde "on" heeft bij de kreatie van een omgevings-frame, wil dit zeggen dat er een bijhorend keuzepunt op de stapel staat.
Als cutflag dan de waarde "off" heeft, wil dit zeggen dat er geen bijhorend keuzepunt op de stapel staat.

Dit houdt in dat een aantal procedurale en index-instructies de waarde van cutflag wijzigen :

```
try-me-else,try                 (kreatie van keuzepunt)
retry-me-else,retry             (updaten van keuzepunt)
                          -> set cutflag ON

call, execute             -> set cutflag OFF

backtrack                 -> set cutflag OFF

allocate                        (kreatie van omgevings-frame)
                          -> KOPIEER cutflag
```

5.2.4. Pseudo-code voor de cut-operatie

```
   while (B <= E) reset(B)
/* B > E */
if ( cutflag on in E )
     { reset(B); set cutflag off in E;};
/* There is a hole in the stack, but that doesn't matter :
   we won't have to move the current omgevings-frame. */
```

5.2.5. Vertalen van cut-operatie

Voorbeeld 1

```
A <- ! .
  . cut-operatie in het lichaam, enkel tijdelijke variabelen
                                 ALLOCATE
                                 .
                                 .
                                 CALLE cut,0
                                 DEALLOCATE
                                 PROCEED
```

Voorbeeld 2

```
A <- ...,!,....
   meer dan 1 oproep in het lichaam
                                 ALLOCATE
                                 .
                                 .
                                 .
                                 CALLE cut,n
                                 .
                                 .
                                 .
```

Voorbeeld 3

```
A <- .....,!.
   meer dan 1 oproep in  het lichaam
                                 ALLOCATE
                                 .
                                 .
                                 .
                                 CALLE cut.0
                                 DEALLOCATE
                                 PROCEED
```

Opdat de cut-operatie correct kan uitgevoerd worden, is het nodig dat er altijd een omgevings-frame dat overeenkomt met de vader-oproep op de stapel staat : eventueel moet het er expliciet opgezet worden (voorbeeld 1), en eventueel moeten wij vermijden dat het omgevings-frame reeds vrijgegeven wordt (voorbeeld 3).

Wij schrijven de pseudo-code uit voor de volgende sekwentie van WIC-instructies :

```
calle cut,0
deallocate
proceed
```

```
P -> calle cut,0
evalpred(cut,0)
P -> deallocate
/* E < B */ A <- E
reset E,CP vanuit E
P ->proceed
P <- CP
```

5.3. Pseudo-code voor proceed

Het volstaat om de programmateller P aan te passen zodat hij naar de continuation wijst. De CP moet hier geen nieuwe waarde krijgen : CP zal altijd eerst nog aangepast worden vooraleer hij opnieuw gebruikt zal worden.

De continuation (CP) verwijst altijd naar :

- ofwel eventueel 1 of meer put-instructies gevolgd door een call-instructie. Deze laatste zorgt voor een nieuwe waarde voor CP.
- ofwel eventueel 1 of meer put-instructies gevolg door een dealex-instructie (samentrekking van deallocate-instructie en execute-instructue). De deallocatie zorgt ervoor dat de waarden van E en CP aangepast worden.

Een CP kan NOOIT verwijzen naar een execute-instructie al dan niet voorafgegaan door put-instructies. Als het lichaam van een proceduredefinitie slechts 1 oproep bevat, zal deze oproep uitgevoerd worden zonder dat de CP wordt gewijzigd :
cfr. de pseudo-code voor execute.

Referentie.

[1] Warren, D.H., An Abstract Prolog Instruction Set, Artificial Intelligence Center, SRI International, August 1983.

Inhoudstafel