

Atelier B

LOGIC-SOLVER

Opérations et gardes

version 1.5



ATELIER B
LOGIC-SOLVER Opérations et gardes
version 1.5

Document établi par CLEARSY.

Ce document est la propriété de CLEARSY et ne doit pas être copié, reproduit, dupliqué
totalement ou partiellement sans autorisation écrite.

Tous les noms des produits cités sont des marques déposées par leurs auteurs respectifs.

CLEARSY
Maintenance ATELIER B
Parc de la Duranne
320 avenue Archimède
Les Pléiades III - Bât.A
13857 Aix-en-Provence Cedex 3
France

Tél 33 (0)4 42 37 12 99
Fax 33 (0)4 42 37 12 71
email : maintenance.atelierb@clearsy.com

Table des matières

1	bUpident	3
2	band	5
3	bappend	7
4	barith	9
5	bcall	11
6	bcall1	15
7	bcatl	19
8	bclean	21
9	bclose	23
10	bcompile	25
11	bconnect	27
12	bcrel	29
13	bcrelr	31
14	bcrer	33
15	bctrule	35
16	bdefl	37
17	bdump	39
18	bflat	41

19 bfork	43
20 bfresh	47
21 bftac	49
22 bfwd	51
23 bget	53
24 bgetd	55
25 bgethyp	57
26 bgetresult	59
27 bgoal	61
28 bguard	63
29 bguardi	67
30 bhalt	71
31 bident	73
32 bidentb	75
33 binhyp	77
34 binter	79
35 bkid	81
36 blemma	83
37 blen	85
38 blenf	87
39 blent	89
40 blident	91
41 blood	93

42 blvar	97
43 bmark	101
44 bmask	103
45 bmatch	105
46 bmodr	107
47 bnewv	109
48 bnfree	111
49 bnlmap	113
50 bnmap	115
51 bnot	117
52 bnum	119
53 bpattern	121
54 bpop	123
55 bprintf	125
56 breade	127
57 breadf	129
58 brecompact	131
59 bresetcomp	133
60 bresult	137
61 brev	139
62 brule	141
63 bsearch	143
64 bsetmode	145

65 bshell	147
66 bslmap	149
67 bsmmap	151
68 bsparemem	153
69 bststatistics	155
70 bstring	157
71 bsubfrm	159
72 btac	161
73 btest	163
74 bunify	165
75 bunmask	167
76 bvrbr	169
77 bwait	173
78 bwritef	175
79 bwriteitem	177

bUpident	3	bnewv	109
band	5	bnfree	111
bappend	7	bnlmap	113
barith	9	bnmap	115
bcall	11	bnot	117
bcall1	15	bnum	119
bcall2	15	bpattern	121
bcat1	19	bpop	123
bclean	21	bprintf	125
bclose	23	breade	127
bcompile	25	breadf	129
bconnect	27	brecompact	131
bcrel	29	bresetcomp	133
bcrelr	31	bresult	137
bcrer	33	brev	139
bctrule	35	brule	141
bdef1	37	bsearch	143
bdef2	37	bsetmode	145
bdump	39	bshell	147
bflat	41	bslmap	149
bfork	43	bsmap	151
bfresh	47	bsparemem	153
bftac	49	bstatistics	155
bfwd	51	bstring	157
bget	53	bsubfrm	159
bgetd	55	btac	161
bgethyp	57	btest	163
bgetresult	59	bunify	165
bgoal	61	bunmask	167
bguard	63	bvrb	169
bguardi	67	bwait	173
bhalt	71	bwritef	175
bident	73	bwritem	177
bidentb	75		
binhyp	77		
binter	79		
bkid	81		
blemma	83		
blen	85		
blenf	87		
blent	89		
blident	91		
bload	93		
blvar	97		
bmark	101		
bmask	103		
bmatch	105		
bmodr	107		

Chapitre 1

bUpident

`bUpident(f)`

Paramètres

f : FORMULE.

Nature

Garde.

Utilisation pratique

Pour tester qu'une formule est un IDENTIFICATEUR.

Évaluation

SUCCES si la formule *f* est un IDENTIFICATEUR constitué de LETTRES majuscules ou du caractère UNDERSCORE. On rappelle qu'un IDENTIFICATEUR est une chaîne de caractères de longueur plus grande que 1, et constituée de LETTRES, de CHIFFRES ou du caractère UNDERSCORE.

Exemple

THEORY ess IS

```
bcall(WRITE: bwritef("test3: ECHEC\n"))
```

```
=>
```

```
test3;
```

```
bUpident(AAA_2) &
```

```
bcall(WRITE: bwritef("test3: SUCCES\n"))
```

```
=>
```

```
test3;
```

```
bcall(WRITE: bwritef("test2: ECHEC\n")) &
```

```
test3
```

```
=>
```

```
test2;
```



```
    bUpident(aaa) &
    bcall(WRITE: bwritef("test2: SUCCES\n")) &
    test3
=>
    test2;

    bcall(WRITE: bwritef("test1: ECHEC\n")) &
    test2
=>
    test1;

    bUpident(AAA) &
    bcall(WRITE: bwritef("test1: SUCCES\n")) &
    test2
=>
    test1;

    bcall(ess: test1)
=>
    coco
```

END

Résultat

```
test1: SUCCES
test2: ECHEC
test3: ECHEC
```

Chapitre 2

band

band(g_1, g_2)

Paramètres

g_1 : GARDE

g_2 : GARDE

Nature

Garde.

Utilisation pratique

Pour coordonner plusieurs gardes.

Évaluation

Plusieurs cas sont à envisager suivant la nature des gardes g_1 et g_2 :

- Si la garde g_1 est la garde **binhyp**(H), ou la garde **binhyp**(n, H), ou la garde **binhyp**(m, n, H), et si, dans les deux derniers cas, n est un JOKER, alors la garde **band** est un SUCCES s'il existe une hypothèse h qui coïncide avec H et qui est telle que la garde g_2 soit un SUCCES.
- Si la garde g_1 est la garde **bsubfrm**, alors la garde **band** est un SUCCES s'il existe une instanciation de jokers par la garde **bsubfrm** (supposée être un SUCCES), qui est telle que la garde g_2 soit un SUCCES.
- Si la garde g_1 est la garde **bsearch**, alors la garde **band** est un SUCCES s'il existe une instanciation de jokers par la garde **bsearch** (supposée être un SUCCES), qui est telle que la garde g_2 soit un SUCCES.
- Si la garde g_1 est la garde **brule**, alors la garde **band** est un SUCCES s'il existe une instanciation de jokers par la garde **brule** (supposée être un SUCCES), qui est telle que la garde g_2 soit un SUCCES.
- Dans tous les autres cas, la garde **band** est un SUCCES si les deux gardes g_1 et g_2 , évaluées l'une après l'autre, sont des SUCCES. En cas d'ECHEC de la seconde, on ne revient pas sur la première comme dans les cas précédents.

Exemple

THEORY ess IS

```
band(binhyp(aaa(x)), band(binhyp(ccc),binhyp(bbb(x))) ) &
bcall(WRITE: bwritef("x: %\n",x))
=>
H;

bcall((DED~;ess):((bbb(1) => (aaa(2) & bbb(2) & ccc & aaa(1) => titi))))
=>
coco

END
```

Résultat

x: 1

Chapitre 3

bappend

`bappend(f)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

Nature

Garde.

Utilisation pratique

Ouverture d'un fichier pour l'opération `bprintf`.

Évaluation

La garde est un `SUCCES` lorsque *f* est un nom de fichier que l'on peut ouvrir en écriture. Si le fichier n'existait pas, il est créé. S'il existait déjà, il n'est pas réinitialisé.

L'opération `bprintf` écrira désormais à la fin du fichier *f* jusqu'à la prochaine exécution d'une garde `bconnect`, d'une garde `bappend` ou d'une opération `bclose`.

Exemple

```
THEORY ess IS
```

```
    bappend("toto") &
    breade("Hello2: ",s) &
    bcall(WRITE: bprintf("Echo2:  %\n",s)) &
    bcall(SHELL: bshell("cat toto"))
=>
    titi;
```

```
    bconnect("toto") &
    breade("Hello1: ",s) &
    bcall(WRITE: bprintf("Echo1:  %\n",s)) &
    bclose &
    titi
=>
```

```
coco
```

```
END
```

Résultat

```
Hello1: Bonjour  
Hello2: Au-revoir  
Echo1:  Bonjour  
Echo2:  Au-revoir
```

Chapitre 4

barith

$$m + n \quad m - n \quad m * n \quad m / n$$

Paramètres

m : NOMBRE

n : NOMBRE

Pré-conditions

L'opération arithmétique spécifiée doit être bien définie, c'est-à-dire produire un nouveau NOMBRE.

Nature

Opération sur un sous-but

Utilisation pratique

Calcul arithmétique sur des constantes numériques.

Évaluation

L'opération arithmétique spécifiée comme suit est effectuée :

- L'opérateur + correspond à l'addition.
- L'opérateur - correspond à la soustraction.
- L'opérateur * correspond à la multiplication.
- L'opérateur / correspond à la division entière.

Tactique

ARI

Exemple

THEORY ess IS

```
bcall(WRITE: bwritef("%\n",P))
```

```
=>
```

```
P;
```

```
bcall( (ARI~;ess): (ff(2+3)+ff(3*2)+ff(3-2+6/3)) )
```

=>

coco

END

Résultat

`ff(5)+ff(6)+ff(3)`

Chapitre 5

bcall

bcall(*l*)

Paramètres

l : LISTE_DE_BUTS_GÉNÉRALISÉS_SÉPARÉS_PAR_DES_BARRES

Un but généralisé est une formule de la forme $t_1, t_2 : f$, ou de la forme $t : f$, avec :

*t*₁ : TACTIQUE_PAR_L'ARRIÈRE

*t*₂ : TACTIQUE_PAR_L'AVANT

t : TACTIQUE_PAR_L'ARRIÈRE

f : FORMULE

Pré-conditions

La formule *f* d'un but généralisé ne doit pas contenir elle-même de **bcall**.

Nature

Opération sur un but.

Utilisation pratique

Pour lancer un nouveau but sur de nouvelles tactiques.

Évaluation

L'évaluation de l'opération **bcall**(*b*₁ | ... | *b*_{*n*}) consiste à suspendre la preuve en cours et à la remplacer par celle du but généralisé *b*₁. En cas d'échec de *b*₁, on effectue un recul, et le but généralisé suivant est essayé, etc ... En cas d'échec du dernier but généralisé *b*_{*n*}, le recul s'effectue jusqu'au plus proche **bcall** non terminé de la preuve qui avait été suspendue. On essaye alors de prouver le but suivant de ce **bcall**, etc ... En cas d'échec final (c'est-à-dire lorsqu'il n'y a plus de **bcall** non terminé), la preuve est arrêtée. Après un succès sur un certain but généralisé, la preuve est reprise au point où elle avait été laissée lors du lancement du **bcall** associé.

La preuve d'un but généralisé de la forme $t_1, t_2 : f$ consiste à prouver la formule *f* à l'aide de la tactique par l'arrière *t*₁ (remplacement des buts par d'autres) et à l'aide de la tactique par l'avant *t*₂ (génération de nouvelles hypothèses). Dans le cas d'un but généralisé de la forme $t : f$, la tactique *t* est une tactique par l'arrière, il n'y a pas de

tactique par l'avant.

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul.

Exemple

THEORY ess IS

```
bcall( (WRITE;ARI;blk)~: prblk(i,"But toto\n") ) &
tata(i+1) &
bcall( (WRITE;ARI;blk)~: prblk(i,"Succes de toto\n") )
=>
toto(i);
```

```
bcall( (WRITE;ARI;blk)~: prblk(i,"But titi\n") ) &
tete(i+1) &
bcall( (WRITE;ARI;blk)~: prblk(i,"Succes de titi\n") )
=>
titi(i);
```

```
bcall( (WRITE;ARI;blk)~: prblk(i,"But tete\n") ) &
beuh(i+1) &
bcall( (WRITE;ARI;blk)~: prblk(i,"Succes de tete\n") )
=>
tete(i);
```

```
bcall( (WRITE;ARI;blk)~: prblk(i,"But tutu\n") ) &
binhyp(aaa) &
bcall( (WRITE;ARI;blk)~: prblk(i,"Succes de tutu\n") )
=>
tutu(i);
```

```
bcall( (WRITE;ARI;blk)~: prblk(i,"But coco\n") ) &
bcall( (ARI;ess)~: toto(i+1) |
      (ARI;ess)~: titi(i+1) |
      (DED;ARI;ess)~,avt: (hyp => tutu(i+1)) ) &
bcall( (WRITE;ARI;blk)~: prblk(i,"Succes de coco\n") )
=>
coco(i);
```

```
bcall(ess: coco(1))
=>
coco
```

END

&

```
THEORY avt IS
```

```
  hyp => aaa
```

```
END
```

```
&
```

```
THEORY blk IS
```

```
  bwritef(" ") &
```

```
  blank(i-1)
```

```
=>
```

```
  blank(i);
```

```
  blank(0);
```

```
  blank(i) &
```

```
  bwritef(f)
```

```
=>
```

```
  prblk(i,f)
```

```
END
```

Résultat

```
But coco
```

```
  But toto
```

```
  But titi
```

```
    But tete
```

```
  But tutu
```

```
  Succes de tutu
```

```
Succes de coco
```


Chapitre 6

bcall1

`bcall1(f)` `bcall2(f)`

Paramètres

f : FORMULE

Nature

Opération sur un but.

Utilisation pratique

Pour appeler le système de trace.

Évaluation

Utilisée dans une règle **Backward**, l'évaluation de l'opération `bcall1(f)` consiste à suspendre la preuve en cours et à la remplacer par celle du but

```
bcall(t : trace(f, but, regle, substitution, theorie.numero))
```

avec :

- **but** : but courant, lors de l'évaluation du `bcall1`,
 - **regle** : corps de la règle appliquée,
 - **substitution** : substitution correspondant à l'instanciation de la règle. Cette substitution est de la forme `[a :=b]`. Dans le cas où la substitution est vide, substitution vaudra `bfalse :=bfalse`
 - **theorie.numero** : position de la règle appelée (**regle**) dans la théorie **theorie**.
- `Sibcall1` est appelé par `bguard(DEF : bcall1(f))`

le but produit sera `bguard(t : trace(f))`

Dans une règle **Forward**, le `bcall1` doit être placé à la fin du conséquent.

La théorie **t** est indiquée par l'opération préalable `bdef1(t)`. Si aucun appel `bdef1` n'a été réalisé ou si le dernier appel était `bdef1(0)`, alors le but `bcall(t : trace(f))` n'est pas évalué et l'opération `bcall1(f)` réussit. Cette opération a été optimisée afin que cette évaluation soit la plus efficace possible.

Le principe de fonctionnement de l'opération `bcall2` (associée à la garde `bdef2`) est iden-

tique à celui de bcall1. **Tactique**

DEF

Exemple

THEORY t1 IS

```
bcall1(regle_t1_avant_bdef) &
bcall(DEF: bdef1(t2)) &
bcall1(regle_t1_apres_bdef_t2)
=>
test1;
```

```
bcall1(regle_t1_apres_bdef_t2bis)
=>
test2;
```

```
bguard(DEF: bcall1(regle_t1_apres_bdef_t2ter)) &
bcall(DEF: bdef1(t3)) &
bcall1(regle_t1_apres_bdef_t3)
=>
test3;
```

```
bcall(DEF: bdef1(0)) &
bcall1(regle_t1_apres_bdef_0)
=>
test4;
```

```
test1 &
test2 &
test3 &
test4
=>
```

P
END

&

THEORY t2 IS

```
bcall(WRITE: bwritef("t2: but = %\n", p))
=>
```

P
END

&

THEORY t3 IS

```
bcall(WRITE: bwritef("t3: but = %\n", p))
=>
```

P
END

Résultat

```
t2: but = trace(regle_t1_apres_bdef_t2bis,test2,
(bcall1(regle_t1_apres_bdef_t2bis) => test2),(bfalse:=bfalse),t1.2)
t2: but = trace(regle_t1_apres_bdef_t2ter)
t3: but = trace(regle_t1_apres_bdef_t3,test3,(bguard(6,0:
bcall1(regle_t1_apres_bdef_t2ter)) & bcall(6,0: bdef1(t3)) &
bcall1(regle_t1_apres_bdef_t3) => test3),(bfalse:=bfalse),t1.3)
```


Chapitre 7

bcatl

`bcatl(l)`

Paramètres

`l` : LISTE_D_ATOMES_OU_DE_CHAINES_ENTRE_GUILLEMETS_SÉPARÉS_DES_VIRGULES.

Nature

Opération sur un sous-but.

Utilisation pratique

Pour construire des chaines entre guillemets.

Évaluation

Les différents constituants de la liste sont concaténés.

Tactique

CATL.

Exemple :

```
THEORY ess IS
```

```
  bcall(WRITE: bwritef("%\n",P))
```

```
=>
```

```
  P;
```

```
  bcall((CATL~;ess): bcatl(aaa,bcatl("/bbb/",ccc),"/",ddd)++bcatl(aa+bb))
```

```
=>
```

```
  coco
```

```
END
```

Résultat

```
"aaa/bbb/ccc/ddd"++bcatl(aa+bb)
```


Chapitre 8

bclean

`bclean(t)`

Paramètres

t : NOM_DE_THÉORIE.

Pré-conditions

La théorie *t* doit exister.

Nature

Opération sur un but

Utilisation pratique

Nettoyage d'une théorie.

Évaluation

Toutes les règles ou lemmes, créés dynamiquement par les opérations `bcrer`, `bcrel` ou `bcrelr` dans la théorie *t*, sont supprimés.

Tactique

RULE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
    blent(beuh.i) &
    bcall(WRITE:bwritef("len3: %\n",i))
=>
    tutu;

    blent(beuh.i) &
    bcall(WRITE:bwritef("len2: %\n",i)) &
    bcall(RULE: bclean(beuh)) &
```

```
tutu
=>
  tata;

  blent(beuh.i) &
  bcall(RULE: (bcrer(beuh,fff) & bcrer(beuh,ggg) & bcrer(beuh,hhh))) &
  bcall(WRITE:bwritef("len1: %\n",i)) &
  tata
=>
  titi;

  bcall(ess: titi)
=>
  coco
```

END

&

THEORY beuh IS

```
aaa;
bbb;
ccc;
ddd;
eee
```

END

Résultat

```
len1: 5
len2: 8
len3: 5
```

Chapitre 9

bclose

bclose

Nature

Opération sur un but

Utilisation pratique

Fermeture d'un fichier.

Évaluation

Le fichier précédemment ouvert par l'intermédiaire d'une garde `bconnect` ou `bappend` est fermé (si aucun fichier n'avaient été ouvert, l'opération n'a pas d'effet). A partir de ce moment, l'opération `bprintf` écrit sur le fichier standard de sortie, et ce jusqu'à ce que d'autres gardes `bconnect` ou `bappend` soient exécutées avec succès.

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
bconnect("toto") &  
bcall(WRITE: bprintf("Bonjour\n")) &  
bclose &  
bcall(WRITE: bprintf("Au-revoir\n"))  
=>  
coco
```

END

Résultat

Au-revoir

Chapitre 10

bcompile

`bcompile(t)`

Paramètres

t : NOM_DE_THÉORIE

Pré-conditions

La théorie *t* doit être *vide*. Pour cela, elle doit nécessairement avoir été créée *vide* dans le fichier source. On peut, cependant, s'en servir dans plusieurs opérations `bcompile` successives, à condition de l'avoir nettoyée, avant chaque usage, à l'aide de l'opération `bclean`. Il est aussi recommandé de nettoyer la mémoire, qui est utilisée par le produit de la compilation, à l'aide de l'opération `bresetcomp`.

Nature

Opération sur un but.

Utilisation pratique

Pour compiler une théorie durant l'exécution.

Évaluation

La théorie *t* est compilée.

Tactique

RULE.

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul.

Exemple

```
THEORY ess IS
```

```
  bget("theo1",R) &  
  bget("theo2",S) &  
  bcrelr(aaa,R) &  
  bcrelr(bbb,S) &
```

```

    bcompile(aaa) &
    bcompile(bbb) &
    bcall((aaa~;bbb~): tutu)

```

```

=>

```

```

    titi;

```

```

    bcall((RULE;ess)~: titi)

```

```

=>

```

```

    coco

```

```

END

```

```

&

```

```

THEORY aaa END

```

```

&

```

```

THEORY bbb END

```

Fichier Theo1

```

    bcall(WRITE:bwritef("Hello_1\n")) &
    toto

```

```

=>

```

```

    tutu;

```

```

    bcall(WRITE:bwritef("Hello_2\n")) &
    tata

```

```

=>

```

```

    toto

```

Fichier Theo2

```

    bcall(WRITE:bwritef("Hello_3\n")) &
    tete

```

```

=>

```

```

    tata;

```

```

    bcall(WRITE:bwritef("Hello_4\n"))

```

```

=>

```

```

    tete

```

Résultat

```

Hello_1

```

```

Hello_2

```

```

Hello_3

```

```

Hello_4

```

Chapitre 11

bconnect

`bconnect(f)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

Nature

Garde.

Utilisation pratique

Ouverture d'un fichier pour l'opération `bprintf`.

Évaluation

La garde est un SUCCES lorsque *f* est un nom de fichier que l'on peut ouvrir en écriture. Si le fichier n'existait pas, il est créé. S'il existait déjà, il est réinitialisé.

L'opération `bprintf` écrira désormais dans le fichier *f* jusqu'à la prochaine exécution d'une garde `bconnect`, d'une garde `bappend` ou d'une opération `bclose`.

Exemple

```
THEORY ess IS

    bconnect("toto") &
    breade("Hello: ",s) &
    bcall(WRITE: bprintf("Echo:  %\n",s)) &
    bcall(SHELL: bshell("cat toto"))
=>
    coco
```

END

Résultat

```
Hello: Bonjour
Echo:  Bonjour
```


Chapitre 12

bcrel

`bcrel(t, r)`

Paramètres

t : NOM_DE_THÉORIE

r : FORMULE

Pré-conditions

Néant.

Nature

Opération sur un but

Utilisation pratique

On utilise cette opération pour créer un nouveau lemme dans une théorie.

Évaluation

Si la théorie *t* n'existe pas, elle est créée. Le lemme *r* est ajoutée à la fin de la théorie *t*. Si, au moment de créer le lemme *r*, on a des hypothèses *H*, le lemme crée est $H \Rightarrow r$. Le nouveau lemme est enregistré *non-prouvé*. De tels lemmes *non-prouvés* peuvent être accédés grâce à la garde **blemma**. On peut aussi accéder à un lemme (qu'il soit prouvé ou non) par la garde **brule**. Un lemme peut être rendu *prouvé* grâce à l'opération **bmark**. Ce lemme peut être supprimé par l'effet des opérations **bpop** ou **bclean**.

Tactique

RULE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
brule(beuh.i,R) &  
bcall(WRITE: bwritef("(%): %\n",i,R))
```

=>

tata;

bcall((DED;RULE;ess)~: (((ccc&ddd&eee) => bcrel(beuh,aaa) & tata))

=>

coco

END

Résultat

ccc & ddd & eee => aaa

Chapitre 13

bcrelr

`bcrelr(t, (l))`

Paramètres

t : NOM_DE_THÉORIE

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_POINTS-VIRGULES

Pré-conditions

Néant.

Nature

Opération sur un but

Utilisation pratique

On utilise cette opération pour créer une suite de règles ou de lemmes.

Évaluation

Si la théorie *t* n'existe pas, elle est créée. On ajoute à la theorie *t*, et dans l'ordre de la liste *l*, autant de règles ou de lemmes qu'il y a de formules dans *l*. Les formules de la forme `bproved(r)` ou `bunproved(r)` donnent des lemmes. Ces lemmes sont créés *proved* ou *unproved* comme il est indiqué. Ces mentions explicites disparaissent lorsque le lemme est ajouté à la théorie (autrement dit, seul le lemme *r* est créé). Les autres formules donnent des règles. Les nouveaux lemmes *non-prouvés* peuvent être accédés grâce à la garde `blemma`. Les règles ou les lemmes peuvent être accédés grâce à la garde `brule`. Un lemme peut être rendu *prouvé* grâce à l'opération `bmark`. Ces lemmes ou règles peuvent être supprimés par l'effet des opérations `bpop` ou `bclean`.

Tactique

RULE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```

    blemma(beuh.i,R) &
    bcall(WRITE: bwritef("(%)\n",i,R))
=>
    toto;

    blent(beuh.i) &
    bcall(WRITE: bwritef("len(beuh): %\n",i)) &
    toto
=>
    tata;

    bcall((DED;RULE): (kkk&iii&jjj =>
                        bcrelr(beuh,(bunproved(aaa);bproved(bbb);ccc)))) &
    tata
=>
    titi;

    bcall(ess~: titi)
=>
    coco

```

END

Résultat

```

len(beuh): 3
(1) aaa

```

Chapitre 14

bcrer

`bcrer(t, r)`

Paramètres

t : NOM_DE_THÉORIE

r : FORMULE

Pré-conditions

Néant.

Nature

Opération sur un but

Utilisation pratique

On utilise cette opération pour créer une nouvelle règle dans une théorie.

Évaluation

Si la théorie *t* n'existe pas, elle est créée. La règle *r* est ajoutée à la fin de la théorie *t*. Si, au moment de créer la règle *r*, on a des hypothèses *H*, la règle effectivement créée est $H \Rightarrow r$. La règle peut être accédée grâce à la garde `brule`. Cette règle peut aussi être supprimée par l'effet des opérations `bpop` ou `bclean`.

Tactique

RULE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
brule(beuh.i,R) &
bcall(WRITE: bwritef("(%): %\n",i,R))
=>
tata;
```

```
bcall((DED;RULE;ess)~: (((ccc&ddd&eee) => bcrer(beuh,aaa) & tata))
=>
titi;

bcall(ess: titi)
=>
coco
```

END

Résultat

```
ccc & ddd & eee => aaa
```

Chapitre 15

bctrule

`bctrule(t , n)`

Paramètres

t : NOM_DE_THEORIE

n : NOMBRE

Nature

Opération sur un but.

Utilisation pratique

Pour mettre à jour les compteurs de règle d'une théorie.

Tactique

RULE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Évaluation

Initialement, les compteurs de règle sont tous égaux à $2^{31} - 1$. L'effet de cette opération est de rendre les compteurs de règle de la théorie *t* tous égaux à *n*. Une règle ne peut être utilisée que si son compteur est positif. Lors de l'utilisation d'une règle, le compteur correspondant est décrémenté.

Exemple

```
THEORY ess IS
```

```
  bcall(WRITE:bwritef("res: %\n",P)) => P;
```

```
  titi(n+2)
=>
  titi(n);
```

```
  bgetallhyp(H) &
```



```

    bcall(WRITE:bwritef("HYP: %\n",H)) &
    bcall(RULE: bctrule(ess,5)) &
    titi(3)
=>
    vovo;

    bcall(RULE: bctrule(tata,10)) &
    bcall((DED;ess),tata~:(aaa & bbb & ccc => vovo))
=>
    coco

END

&

THEORY tata IS

    aaa => eee & eee(aa);

    eee(a) => eee(a+1)/* & bwritef("cccccccc\n")*/;

    bbb & aaa & eee => kkk

END

```

Résultat

```

HYP: aaa & bbb & ccc & eee & eee(aa) & kkk & eee(aa+1) & eee(aa+1+1) &
eee(aa+1+1+1) & eee(aa+1+1+1+1) & eee(aa+1+1+1+1+1)
res: titi(3+2+2+2+2+2)

```

Chapitre 16

bdef1

`bdef1(t)` `bdef2(t)`

Paramètres

`t` : THÉORIE

Nature

Opération sur un but.

Utilisation pratique

Pour déterminer la théorie qui sera utilisée lors de l'appel au système de trace.

Tactique

DEF

Évaluation

Cette opération permet de déterminer la théorie qui sera utilisé lors de l'appel au système de trace (on ne peut pas utiliser une tactique). `bdef1` permet de paramétrer l'appel `bcall1`, et `bdef2` permet de paramétrer l'appel `bcall2`.

Par exemple, après un `bdef1(t)`, l'opération `bcall1(b)` produira le but `bcall(t : trace(b, but, regle, substitution, theorie.numero))`.

La désactivation se réalise grâce à `bdef1(0)` (ou `bdef2(0)`). Dans ce cas, `bcall1` (ou `bcall2`) termine immédiatement, avec succès.

Exemple

```
THEORY t1 IS
```

```
bcall1(truc)
```

```
=>
```

```
test1;
```

```
bcall(DEF: bdef1(t2)) &
test1 &
bcall(DEF: bdef1(t3)) &
test1
=>
p
END
&
THEORY t2 IS
bcall(WRITE: bwritef("t2: but = %\n", p))
=>
p
END
&
THEORY t3 IS
bcall(WRITE: bwritef("t3: but = %\n", p))
=>
p
END
```

Résultat

```
t2: but = trace(truc,test1,(bcall1(truc) => test1),(bfalse:=bfalse),t1.1)
t3: but = trace(truc,test1,(bcall1(truc) => test1),(bfalse:=bfalse),t1.1)
```

Chapitre 17

bdump

`bdump(l, f)`

Paramètres

l : LISTE_DE_NOM_DE_THÉORIE_SÉPARÉS_PAR_DES_VIRGULES

f : CHAÎNE_ENTRE_GUILLEMETS

Pré-conditions

f et *f.sym* doivent être des fichiers sur lesquels on a le droit d'écrire.

Nature

Opération sur un but.

Utilisation pratique

Pour enregistrer le contenu de certaines théories sur un fichier binaire.

Évaluation

Le contenu de chacune des théories existantes et non vides, mentionnées dans la liste *l*, est stocké en format binaire sur le fichier *f*. Un extrait de la table des symboles correspondants est stocké dans un fichier de nom *f.sym*. Ces deux fichiers peuvent être relus par l'opération `bload`.

Une règle *r* qui est un lemme prouvé est stockée sous la forme `bproved(r)`. Une règle *r* qui est un lemme non prouvé est stockée sous la forme `bunproved(r)`.

Tactique

RULE.

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul.

Exemple

THEORY ess IS

```
bcall(RULE: bdump(aaa,bbb,ccc,"toto"))  
=>
```

```
    titi;

    bcall(ess:titi)
=>
    coco

END

&

THEORY aaa IS

    bunproved(aaa1)

END

&

THEORY bbb IS

    bunproved(bbb1);
    bproved(bbb2)

END

&

THEORY ccc IS

    ccc1;
    ccc2;
    ccc3

END
```

Résultat

Fichiers `toto` et `toto.sym` (voir l'opération `bload`).

Chapitre 18

bflat

`bflat(l)`

Paramètres

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_V_PV_A

Nature

Opération sur un sous-but.

Utilisation pratique

Applatissage d'une liste.

Évaluation

Applatire (récursivement) la liste *l*.

Tactique

FLAT.

Exemple

```
THEORY ess IS
```

```
  bcall(WRITE: bwritef("res: %\n",P))
```

```
=>
```

```
  P;
```

```
  bcall((FLAT;ess): (fff+++bflat(aaa,(bbb,(cc1,cc2,cc3),ddd))+++eee))
```

```
=>
```

```
  coco
```

```
END
```

Résultat

```
fff+++ (aaa,bbb,cc1,cc2,cc3,ddd)+++eee
```


Chapitre 19

bfork

bfork(*l*)

Paramètres

l : LISTE_D_ATOMES_OU_DE_CHAINES_ENTRE_GUILLEMETS_SÉPARÉS_DES_VIRGULES.

Nature

Garde.

Utilisation pratique

Pour lancer une commande extérieure.

Évaluation

Les éléments de la liste *l* sont concaténés entre eux, en ajoutant des espaces de séparation et en transformant les atomes en chaînes entre guillemets. Si la liste *l* contient autre chose que des atomes ou des chaînes entre guillemets, la garde échoue.

La chaîne ainsi constituée correspond à la commande extérieure qui est lancée. L'exécution de la règle courante est suspendue jusqu'au moment où cette commande se termine : la garde est alors un succès si l'exécution est correcte.

ATTENTION : si la commande n'est pas trouvée ou ne correspond pas à un fichier exécutable, le LOGIC SOLVER produit un message d'erreur en plus de l'échec de la garde.

Description pour UNIX

Sur UNIX, la commande est lancée en tant que tâche fille du LOGIC SOLVER. On ne lance pas de **shell** intermédiaire, donc la commande ‘**ls ;ls**’ par exemple, ne serait pas correcte puisque le point virgule correspond à une fonctionnalité du **shell**. La commande fille hérite de tout l'environnement du LOGIC SOLVER, en particulier la recherche de l'exécutable ou du script à lancer se fait avec le **path** courant. La garde **bfork** est un succès si la commande a pu s'exécuter et qu'elle retourne un **status** nul.

Si le LOGIC SOLVER reçoit un signal SIGTERM durant l'exécution d'un **bfork**, le signal est retransmis à la tâche fille, sauf s'il est masqué (voir garde **bmask**). L'exécution de la commande se poursuit. Dans la plupart des cas, la tâche fille interprète SIGTERM comme un signal d'arrêt et termine ; le LOGIC SOLVER poursuit alors dans les conditions normales après réception d'un SIGTERM (garde **binter** activée, sous preuve lancée par

bguardi éventuelle arrêtée). Si le signal est masqué, le signal est mémorisé mais il n'est pas transmis à la tâche fille. Il n'est pris en compte que lors de la prochaine garde `bunmask`, le `bfork` est forcément terminé.

Exemple

THEORY ess2 IS

```

p;

bcall(WRITE: bwritef("echec commande sleep 5\n")) &
coco7
=>
coco6;

bguard(WRITE: bwritef("attente 5s...\n")) &
bfork("sleep 5") &
coco7
=>
coco6;

bcall(WRITE: bwritef("echec normal commande badret : retourne status != 0\n")) &
coco6
=>
coco5;

bfork("badret") &
coco6
=>
coco5;

bcall(WRITE: bwritef("echec normal commande sdfs\n")) &
coco5
=>
coco4;

bfork("sdfs") &
coco5
=>
coco4;

bcall(WRITE: bwritef("echec normal commande echo aa+bb\n")) &
coco4
=>
coco3;

bfork("echo", (aa+bb)) &
coco4
=>

```

```

coco3;

bcall(WRITE: bwritef("echec normal commande echo a\n")) &
coco3
=>
coco2;

bfork("echo",a) &
coco3
=>
coco2;

band(bsubfrm(x,toto,(xx+yy+zz),Q),
band(bnot(bpattern(x,(a+b))),
band(bfork("echo",x),(xx\xx)) ))
=>
coco2;

bcall(WRITE: bwritef("echec commande echo toto titi tata separee\n")) &
coco2
=>
coco1;

bfork("echo",toto,"titi tata") &
coco2a
=>
coco1;

bcall(WRITE: bwritef("echec commande echo toto\n")) &
coco1
=>
coco;

bfork("echo toto") &
coco1
=>
coco

END

```

Dans cet exemple, **badret** est un exécutable qui retourne un code d'erreur.

Résultat

```

krt -c ess2
krt -b ess2.kin ess.ex
toto
toto titi tata
zz

```

```
yy
xx
echec normal commande echo a
echec normal commande echo aa+bb
cannot exec sdf$ (probably not found or no execute permission)
echec normal commande sdf$
echec normal commande badret : retourne status != 0
attente 5s...
```

Chapitre 20

bfresh

`bfresh(v, f, V)`

Paramètres

v : VARIABLE

f : FORMULE

V : JOKER

Nature

Garde.

Utilisation pratique

Obtenir des variables fraîches.

Évaluation

La garde est toujours un SUCCES. Le JOKER V est instancié avec autant de variables que les variables de v . De plus, les nouvelles variables V ne sont pas libres dans f . Si v n'est pas libre dans f , alors V est identique à v .

Exemple

```
THEORY ess IS

    bfresh((xx,yy,zz),aaa+xx$0-yy*zz,V) &
    bcall(WRITE: bwritef("%\n",V))
=>
    coco

END
```

Résultat

```
xx$1,yy$1,zz$1
```


Chapitre 21

bftac

`bftac(t, f)`

Paramètres

t : NOM_DE_THÉORIE

f : FORMULE

Nature

Opération sur un but.

Utilisation pratique

Pour créer une théorie.

Évaluation

Le deuxième paramètre *f* ne sert à rien. L'effet de cette opération est de créer la théorie *t*, si elle ne l'était déjà.

Tactique

TACTIC.

Recul

L'effet de bord de cette opération n'est pas détruit lors d'un recul.

Exemple

```
THEORY ess IS
```

```
    blent(aaa.n) &
    bcall(WRITE: bwritef("Taille de aaa: %\n",n))
=>
    toto;

    bftac(aaa,1) &
    bcall(WRITE: bwritef("Creation de aaa\n")) &
    toto
=>
```

```
titi;

blent(aaa.n) &
bcall(WRITE: bwritef("taille de aaa: %\n",n))
=>
titi;

bcall((DED;TACTIC;ess)~: titi)
=>
coco

END
```

Résultat

```
Creation de aaa
Taille de aaa: 0
```

Chapitre 22

bfwd

$\text{bfwd}(f)$

Paramètres

f : FORMULE

Nature

Opération sur un but.

Utilisation pratique

Pour déclencher la tactique forward.

Évaluation

Lorsque la formule f est de la forme $g \Rightarrow h$, cette opération a pour effet de rendre h le but courant après avoir déclenché les tactiques “forward” sur chacune des composantes conjonctives de g . A noter que les composantes conjonctives de g ne sont pas entrées comme nouvelles hypothèses, qu’elles soient ou non déjà des hypothèses.

Lorsque la formule f n’est pas de la forme $g \Rightarrow h$, cette opération a pour effet de rendre f le but courant.

Exemple

```
THEORY ess IS

  bgetallhyp(H) &
  bcall(WRITE: bwritef("hyp2: %\n",H))
=>
  toto;

  bcall(WRITE: bwritef("hyp1: %\n",H)) &
  bcall(ess~,fwd: bfwd(aaa & bbb => toto))
=>
  titi;

  bcall((DED;ess~): (aaa & bbb & ccc => titi))
```



```
=>
  coco
```

```
END
```

```
&
```

```
THEORY fwd IS
```

```
  aaa => ggg;
  bbb & ccc & ggg => kkk
```

```
END
```

Résultat

```
hyp1: aaa & bbb & ccc
hyp2: aaa & bbb & ccc & ggg & kkk
```

Chapitre 23

bget

$\text{bget}(f, g)$

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

g : FORMULE

Nature

Garde.

Utilisation pratique

Rentrer une information depuis un fichier texte.

Évaluation

La garde est un SUCCES lorsque la chaîne de caractère f correspond à un nom de fichier avec la permission en lecture et lorsque le contenu du fichier coïncide avec la formule g . Les jokers non instanciés de g sont instanciés.

Exemple

```
THEORY ess IS

  bget("toto", (A-B&C)) &
  bcall(WRITE: bwritef("% % % \n", A, B, C))
=>
  titi;

  bcall(ess: titi)
=>
  coco

END
```

Résultat

Bonjour Monsieur Madame

Chapitre 24

bgetd

`bgetd(f, g)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

g : FORMULE

Nature

Garde.

Utilisation pratique

Rentrer une information depuis un fichier texte, sans utilisation de *cpp* (préprocesseur).

Évaluation

La garde est un SUCCES lorsque la chaîne de caractère *f* correspond à un nom de fichier avec la permission en lecture et lorsque le contenu du fichier coïncide avec la formule *g*. Les jokers non instanciés de *g* sont instanciés.

Le contenu du fichier n'est pas traité par *cpp*, ce qui permet de rendre le programme plus rapide et de nécessiter moins de mémoire (il n'y a pas de duplication par **fork**). En particulier, toutes les macros (ex : **#include**, **#define**, ...) ne sont pas prises en compte.

Exemple

THEORY toto IS

```
bgetd("toto",F) &  
bcall(WRITE: bwritef("should be aa+cc+dd : %\n",F))  
=>  
coco1;
```

```
bgetd("titi",F) &  
bcall(WRITE: bwritef("comportement anormal du bgetd\n"))  
=>  
coco1;
```

```
bget("titi",F) &
bcall(WRITE: bwritef("should give missing symbol between toto and titi\n")) &
coco1
=>
coco
```

END

avec fichier titi :

```
#define toto titi+
toto titi
```

et fichier toto :

```
aa+cc+dd
```

Résultat

```
should give missing symbol between toto and titi
line 2: missing binary symbol between toto and titi
should be aa+cc+dd : aa+cc+dd
```

Chapitre 25

bgethyp

`bgethyp(h)` `bgetallhyp(h)`

Paramètres

h : FORMULE

Nature

Garde.

Utilisation pratique

Obtenir les hypothèses d'une preuve.

Évaluation

La garde est un SUCCES lorsque la preuve comporte des hypothèses et que la conjonction de ces hypothèses coïncide avec la formule *h*. Les hypothèses concernées dépendent de la garde :

- Avec la garde `bgethyp`, on n'obtient que les hypothèses principales.
- Avec la garde `bgetallhyp` on obtient les hypothèses principales ainsi que les hypothèses dérivées de ces hypothèses principales.

Dans tous les cas, les jokers non instanciés de *h* sont instanciés.

Exemple

```
THEORY ess IS
```

```
  bgethyp(H) &
  bgetallhyp(G) &
  bcall(WRITE: bwritef("Hypotheses principales:      %\n",H)) &
  bcall(WRITE: bwritef("Hypotheses princ. et derivees: %\n",G))
=>
  P;

  bcall((DED;ess),fwd: (aaa & bbb => ggg))
=>
  coco
```

END

&

THEORY fwd IS

aaa => ccc;

ccc & bbb => ddd & eee

END

Résultat

Hypotheses principales: aaa & bbb

Hypotheses princ. et derivees: aaa & bbb & ccc & ddd & eee

Chapitre 26

bgetresult

`bgetresult(f)`

Paramètres

f : FORMULE

Nature

Garde.

Utilisation pratique

Pour avoir accès au résultat stocké par l'opération `bresult`.

Évaluation

Pour que la garde soit un SUCCES il faut d'abord qu'il existe un résultat effectivement stocké : on doit donc être sous le contrôle d'une garde `bguard` et une opération `bresult` a du été exécutée. Il faut ensuite que le résultat stocké coïncide avec la formule *f*.

Les jokers non instanciés de *f* sont instanciés dans la suite de la règle.

Exemple

```
THEORY ess IS
```

```
  bguard((srt;RES)~: sort(aaa & bbb & bbb & aaa & aaa & bbb & bbb),(r&t)) &  
  bcall((FLAT;WRITE): bwritef("res: %\n",bflat(r&t)))  
=>  
  coco
```

```
END
```

```
&
```

```
THEORY srt IS
```

```
  bgetresult(a&b) &
```



```
bresult(a&(b&bbb))
```

```
=>
```

```
bbb;
```

```
bgetresult(a&b) &
```

```
bresult(a&aaa&b)
```

```
=>
```

```
aaa;
```

```
bgetresult(a&?) &
```

```
bresult(a&(bbb))
```

```
=>
```

```
bbb;
```

```
bgetresult(?&b) &
```

```
bresult(aaa&b)
```

```
=>
```

```
aaa;
```

```
bresult(? & ?) &
```

```
x
```

```
=>
```

```
sort(x)
```

```
END
```

Résultat

```
res: aaa & aaa & aaa & bbb & bbb & bbb & bbb
```

Chapitre 27

bgoal

`bgoal(f)`

Paramètres

f : FORMULE

Nature

Garde.

Utilisation pratique

Pour obtenir la forme du but courant.

Évaluation

La garde est un SUCCES si le but courant coïncide avec *f*. Les jokers non instanciés de *f* sont instanciés.

Exemple

```
THEORY ess IS

  bgoal(P) &
  bcall(WRITE: bprintf("goal1: %\n",P)) &
  bcall(titi~: aa+bb-cc)
=>
  coco

END

&

THEORY titi IS

  bcall(WRITE: bprintf("res:  % \n",P))
=>
  P;
```

```
bgoal(P-Q) &  
bcall(WRITE: bprintf("goal2: % %\n",P,Q))  
=>  
a+b == a-b  
  
END
```

Résultat

```
goal1: coco  
goal2: aa+bb cc  
res:   aa-bb-cc
```

Chapitre 28

bguard

$\text{bguard}(t : f, g)$ $\text{bguard}(t : h)$

Paramètres

t : LISTE_DE_DEUX_TACTIQUES_AU_PLUS__SÉPARÉES_PAR_UNE_VIRGULE

Une LISTE_DE_DEUX_TACTIQUES_AU_PLUS__SÉPARÉES_PAR_UNE_VIRGULE est soit de la forme t_1, t_2 , soit de la forme t_1 , avec :

t_1 : TACTIQUE_PAR_L'ARRIÈRE

t_2 : TACTIQUE_PAR_L'AVANT

f : FORMULE

g : FORMULE

h : FORMULE_SANS_VIRGULE_AU_PLUS_HAUT_NIVEAU

Pré-conditions

Les formules f , g ou h ne doivent pas contenir de `bcall` ou de `bguard`.

Nature

Garde.

Utilisation pratique

Pour pouvoir programmer une garde explicitement.

Évaluation

L'évaluation de cette garde s'effectue d'abord en lançant la preuve du but f (première forme) ou du but h (deuxième forme) avec la tactique par l'arrière t_1 et, éventuellement, la tactique par l'avant t_2 .

La garde $\text{bguard}(t : f, g)$ est un SUCCÈS si la preuve de f réussit et si le “résultat” (voir l'opération `bresult`) de cette preuve existe et coïncide avec le pattern g . Les JOKERS non instanciés de g sont instanciés.

La garde $\text{bguard}(t : h)$ est un SUCCÈS si la preuve de h réussit.

Exemple

```
#define INTRP1(a,r)  bguard(((ARI;int1)~;RES): intrp(a),r)
```

```

#define INTRP2(a,r)  bguard((int2;RES): intrp(a),r)
#define INTRP3(a,r)  bguard((int3;RES): intrp(a),r)

THEORY ess0 IS

    INTRP2(7*3+(2*6)/4-50,r) &
    bcall((FLAT;WRITE): bwritef("res: %\n",bflat(r))) &
    INTRP3(r,s) &
    bcall(WRITE: bwritef("res: %\n",s)) &
    INTRP1(s,t) &
    bcall(WRITE: bwritef("res: %\n",t))
=>
    coco

END

&

THEORY int1 IS

    bresult(UNDEFINED)
=>
    intrp(a);

    bnum(a) &
    bresult(a)
=>
    intrp(a);

    INTRP1(a,r) &
    INTRP1(b,s) &
    intrp(r+s)
=>
    intrp(a+b);

    INTRP1(a,r) &
    INTRP1(b,s) &
    intrp(r*s)
=>
    intrp(a*b);

    INTRP1(a,r) &
    INTRP1(b,s) &
    intrp(r/s)
=>
    intrp(a/b);

    INTRP1(a,r) &

```

```

    INTRP1(b,s) &
    intrp(r-s)
=>
    intrp(a-b)

END

&

THEORY int2 IS

    bresult(a)
=>
    intrp(a);

    INTRP2(a,r) &
    INTRP2(b,s) &
    bresult(plus,r,s)
=>
    intrp(a+b);

    INTRP2(a,r) &
    INTRP2(b,s) &
    bresult(mult,r,s)
=>
    intrp(a*b);

    INTRP2(a,r) &
    INTRP2(b,s) &
    bresult(div,r,s)
=>
    intrp(a/b);

    INTRP2(a,r) &
    INTRP2(b,s) &
    bresult(moins,r,s)
=>
    intrp(a-b)

END

&

THEORY int3 IS

    bresult(a)
=>

```

```
    intrp(a);

    INTRP3(a,r) &
    INTRP3(b,s) &
    bresult(r+s)
=>
    intrp(plus,a,b);

    INTRP3(a,r) &
    INTRP3(b,s) &
    bresult(a*b)
=>
    intrp(mult,a,b);

    INTRP3(a,r) &
    INTRP3(b,s) &
    bresult(r/s)
=>
    intrp(div,a,b);

    INTRP3(a,r) &
    INTRP3(b,s) &
    bresult(r-s)
=>
    intrp(moins,a,b)

END
```

Résultat

```
res: moins,plus,mult,7,3,div,mult,2,6,4,50
res: 7*3+2*6/4-50
res: UNDEFINED
```

Chapitre 29

bguardi

bguardi($t : f, g$) **bguard**($t : h$)

Paramètres

t : LISTE_DE_DEUX_TACTIQUES_AU_PLUS__SÉPARÉES_PAR_UNE_VIRGULE

Une LISTE_DE_DEUX_TACTIQUES_AU_PLUS__SÉPARÉES_PAR_UNE_VIRGULE est soit de la forme t_1, t_2 , soit de la forme t_1 , avec :

t_1 : TACTIQUE_PAR_L'ARRIÈRE

t_2 : TACTIQUE_PAR_L'AVANT

f : FORMULE

g : FORMULE

h : FORMULE_SANS_VIRGULE_AU_PLUS_HAUT_NIVEAU

Pré-conditions

Les formules f , g ou h ne doivent pas contenir de **bcall** ou de **bguard**.

Nature

Garde.

Utilisation pratique

Même utilisation que **bguard**, mais la sous preuve lancée par **bguardi** peut être mise artificiellement en échec par un signal extérieur. Sous *UNIX*, le signal permettant d'interrompre est **SIGTERM**.

Évaluation

L'évaluation de cette garde s'effectue exactement comme un **bguard** normal, mais la réception d'un signal extérieur choisi provoque l'échec de la sous preuve, et donc l'évaluation à **FALSE**. Quand le signal arrive, plus aucune règle ne s'applique (les coïncidences de formules sont considérées comme toujours fausses) jusqu'au retour dans le **bguardi** dernièrement lancé. Le **bguardi** est évalué à **FALSE** et la preuve continue normalement. Le signal n'a aucun effet si il est détecté hors d'un **bguardi**.

Sur *UNIX*, le signal choisi est **SIGTERM**.

Exemple

THEORY ess IS

```

bcall(WRITE: bwritef("interrompu\n"))
=>
glop;

bguardi(ess1: toto) &
bcall(WRITE: bwritef("non interrompu\n"))
=>
glop;

bkid(n) &
bconnect("toto") &
bcall(WRITE: bprintf("%", n)) &
bclose &
glop
=>
glip

```

END

&

THEORY ess1 IS

```

ff(P)
=>
P

```

END

Résultat

```

mach% kill -TERM 'cat toto'
interrompu

```

Le pid de l'exécutable est obtenu grâce à la commande `bkid` et imprimé dans `toto`. L'envoi du signal permet alors d'interrompre la théorie `ess1` qui boucle et de passer à la règle suivante qui imprime `interrompu`.

Autre exemple

THEORY ess0 IS

```

bkid(n) &
bconnect("toto") &
bcall(WRITE: bprintf("%", n)) &
bclose &

```

```
    bcall((DED;ess1~): (tata & toto & titi => glip))
=>
P

END

&

THEORY ess1 IS

    band(binhyp(h),
    bguardi(ess2: hh(h)))
=>
    glip

END

&

THEORY ess2 IS

    gg(gg(x))
=>
    gg(x);

    gg(toto)
=>
    ff(toto);

    ff(tata);

    bguardi(ess3: plouf)
=>
    ff(tata);

    bcall(WRITE: bwritef("essai sur hyp %\n", P)) &
    ff(P)
=>
    hh(P)

END

&

THEORY ess3 IS

    ff(p)
=>
```

p

END

Résultat

```
essai sur hyp titi
essai sur hyp toto
mach% kill -TERM 'cat toto'
essai sur hyp tata
mach% kill -TERM 'cat toto'
```

Les trois hypothèses `titi` `toto` `tata` sont traitées dans cet ordre par le `band`. `titi` provoque un échec, donc on passe à l'hypothèse suivante : `toto` qui provoque un bouclage. L'opérateur débloque la preuve par un signal, la preuve continue par le traitement de `tata`. Celle ci provoque un bouclage dans un deuxième `bguardi` : L'opérateur débloque la deuxième sous preuve par un signal. La règle `ff(tata)` s'applique et la preuve se termine.

Chapitre 30

bhalt

bhalt

Nature

Opération sur un but.

Utilisation pratique

Pour arrêter proprement l'exécution.

Évaluation

L'exécution est arrêtée et les théories sont nettoyées (`bclean` sur toutes les théories).

Exemple

```
THEORY ess IS

  bcall(WRITE: bwritef("Bonjour\n")) &
  bhalt &
  bcall(WRITE: bwritef("Au-revoir\n"))
=>
  coco

END
```

Résultat

Bonjour

Chapitre 31

bident

`bident(f)`

Paramètres

f : FORMULE.

Nature

Garde.

Utilisation pratique

Pour tester qu'une formule est un IDENTIFICATEUR.

Évaluation

SUCCES si la formule *f* est un IDENTIFICATEUR. On rappelle qu'un IDENTIFICATEUR est une chaîne de caractères de longueur plus grande que 1, et constituée de LETTRES, de CHIFFRES ou du caractère UNDERSCORE.

Exemple

THEORY ess IS

```
bcall(WRITE: bwritef("test3: ECHEC\n"))
```

```
=>
```

```
test3;
```

```
bident(a+b) &
```

```
bcall(WRITE: bwritef("test3: SUCCES\n"))
```

```
=>
```

```
test3;
```

```
bcall(WRITE: bwritef("test2: ECHEC\n")) &
```

```
test3
```

```
=>
```

```
test2;
```

```
    bident(a) &
    bcall(WRITE: bwritef("test2: SUCCES\n")) &
    test3
=>
    test2;

    bcall(WRITE: bwritef("test1: ECHEC\n")) &
    test2
=>
    test1;

    bident(aaaa_3) &
    bcall(WRITE: bwritef("test1: SUCCES\n")) &
    test2
=>
    test1;

    bcall(ess: test1)
=>
    coco
```

END

Résultat

```
test1: SUCCES
test2: ECHEC
test3: ECHEC
```

Chapitre 32

bidentb

`bidentb(f)`

Paramètres

f : FORMULE.

Nature

Garde.

Utilisation pratique

Pour tester qu'une formule est un IDENTIFICATEUR B.

Évaluation

SUCCES si la formule *f* est un IDENTIFICATEUR B. On rappelle qu'un IDENTIFICATEUR B est une suite de caractères de longueur plus grande que 1, constituée de LETTRES, de CHIFFRES ou du caractère UNDERSCORE, et commençant obligatoirement par un caractère.

Exemple

THEORY glop IS

```
bnot(bidentb("toto")) &  
bcall(WRITE: bwritef("\"toto\" is not a B identifier\n"))  
=>  
b5;
```

```
bnot(bidentb(s)) &  
bcall(WRITE: bwritef("s is not a B identifier\n")) &  
b5  
=>  
b4;
```

```
bnot(bidentb(8toto)) &  
bcall(WRITE: bwritef("8toto is not a B identifier\n")) &
```



```
b4
=>
b3;

bnot(bidentb(_toto)) &
bcall(WRITE: bwritef("_toto is not a B identifier\n")) &
b3
=>
b2;

bidentb(toto) &
bcall(WRITE: bwritef("toto is a B identifier\n")) &
b2
=>
b1

END
```

Résultat

```
toto is a B identifier
_toto is not a B identifier
8toto is not a B identifier
s is not a B identifier
"toto" is not a B identifier
```

Chapitre 33

binhyp

$\text{binhyp}(h)$ $\text{binhyp}(n, h)$ $\text{binhyp}(m, n, h)$

Paramètres

h : FORMULE

n : FORMULE

m : NOMBRE

Nature

Garde.

Utilisation pratique

Pour accéder à une hypothèse.

Évaluation

La garde $\text{binhyp}(h)$ est un SUCCES lorsqu'il existe une hypothèse qui coïncide avec la formule h . Lorsqu'il y en a plusieurs, on choisit la dernière d'entre elles. Les JOKERS non instanciés de h sont instanciés.

Pour la garde $\text{binhyp}(n, h)$, on considère différents cas suivant la nature de n .

- Lorsque n est un NOMBRE, la garde est un SUCCES si l'hypothèse de rang n existe et coïncide avec la FORMULE h . Les JOKERS non instanciés de h sont instanciés.
- Lorsque n est un JOKER, la garde est un SUCCES s'il existe une hypothèse qui coïncide avec la FORMULE h . Lorsqu'il y a plusieurs hypothèse qui coïncident avec h , on choisit la dernière d'entre elles. Le JOKER n est alors instancié avec le numéro de l'hypothèse sélectionnée. Les JOKERS non instanciés de h sont instanciés.
- Dans tous les autres cas, la garde est un ECHEC.

L'évaluation de la garde $\text{binhyp}(i, n, h)$ correspond à ce que nous venons d'expliquer pour la garde $\text{binhyp}(n, h)$, à ceci près que, dans le cas où n est un JOKER, l'hypothèse sélectionnée est la dernière hypothèse, qui coïncide avec h , et dont le numéro est inférieur ou égal à i .

On notera la grande ressemblance entre ces deux dernières formes de la garde binhyp et les gardes brule et blemma .

Exemple

```

THEORY ess IS

  toto(i);

  binhyp(i,n,H) &
  bcall(WRITE: bwritef("hyp_%: %\n",n,H)) &
  toto(n-1)
=>
  toto(i);

  binhyp(n,H) &
  bcall(WRITE: bwritef("hyp_%: %\n",n,H)) &
  toto(n-1)
=>
  tata;

  binhyp(1,H) &
  bcall(WRITE: bwritef("hyp_1: %\n",H)) &
  tata
=>
  titi;

  binhyp(H+G) &
  bcall(WRITE: bwritef("hyp: %\n",H+G)) &
  titi
=>
  P;

  bcall((DED;(ARI;ess)~):((aaa & bbb+ccc & ddd) => pp))
=>
  coco

END

```

Résultat

```

hyp:   bbb+ccc
hyp_1: aaa
hyp_3: ddd
hyp_2: bbb+ccc
hyp_1: aaa

```

Chapitre 34

binter

binter

Paramètres

Aucun paramètre.

Nature

Garde.

Utilisation pratique

Tester si le *LOGIC-SOLVER* a détecté un signal.

Évaluation

La garde est un SUCCES lorsque le *LOGIC-SOLVER* a détecté un signal choisi. L'évaluation de cette garde remet à zéro l'indicateur de signal détecté.

Il n'y a qu'un seul indicateur : il est impossible de savoir si on a détecté plus d'un signal avant l'évaluation. Sous *UNIX*, le signal reconnu est SIGTERM.

Exemple

THEORY ess IS

```
ff(p)
=>
p;

bcall((DED;ess1~): (aa & bb => glip))
=>
glop;

binter &
bcall(WRITE: bwritef("premiere interruption recue\n")) &
glop
=>
p;
```

```
bkid(n) &
bconnect("toto") &
bcall(WRITE: bprintf("%", n)) &
bclose &
glap
=>
sdfsd
```

END

&

THEORY ess1 IS

```
ff(p)
=>
p;

band(binhyp(aa), binter) &
bcall(WRITE: bwritef("deuxieme interruption recue\n"))
=>
p
```

END

Résultat

```
mach% krt -b ess.kin ess.ex
mach% kill -HUP 'cat toto'
mach% kill -TERM 'cat toto'
premiere interruption recue
mach% kill -TERM 'cat toto'
deuxieme interruption recue
mach%
```

Chapitre 35

bkid

`bkid(f)`

Paramètres

f : FORMULE

Nature

Garde.

Utilisation pratique

Obtenir l'identifiant par lequel le *LOGIC-SOLVER* est connu du système d'exploitation, pour pouvoir lui envoyer des signaux (voir `bguardi`). `bkid` signifie *KernelIDentifier*.

Évaluation

La garde est un *SUCCES* lorsque l'identifiant par lequel le *LOGIC-SOLVER* est connu du système d'exploitation coïncide avec la formule *f*. Les jokers non instanciés de *f* sont instanciés. Sous *UNIX*, l'identifiant est le `pid` de la tâche : c'est un numéro.

Exemple

se reporter à l'exemple de `bguardi`.

Chapitre 36

blemma

`blemma(t.n, f)`

`blemma(m, t.n, f)`

Paramètres

t : NOM_DE_THEORIE

n : FORMULE

f : FORMULE

m : NOMBRE

Nature

Garde.

Utilisation pratique

Rechercher un lemme dans une théorie.

Évaluation

Cette garde fonctionne exactement comme la garde **brule**, à ceci près que la recherche ne s'effectue plus sur des règles quelconques mais uniquement sur des lemmes *non prouvés*. De tels lemmes peuvent être créés au moyen des opérations **bcrel** ou **bcrelr**. Ou encore avoir été placés initialement dans une théorie.

Exemple

```
THEORY ess IS
```

```
  bcall(WRITE: bwritef("test4: ECHEC\n"))
```

```
=>
```

```
  test4;
```

```
  blemma(test.j,r-s) &
```

```
  bcall(WRITE: bwritef("test4: (%) % \n",j,r-s))
```

```
=>
```

```
  test4;
```

```
  blemma(test.5,r) &
```



```

    bcall(WRITE: bwritef("test3: (5) %\n",r)) &
    test4
=>
    test3;

    blemma(3,test.j,r) &
    bcall(WRITE: bwritef("test2: (%) %\n",j,r)) &
    test3
=>
    test2;

    blemma(test.j,r+s) &
    bcall(WRITE: bwritef("test1: (%) % \n",j,r+s)) &
    test2
=>
    test1;

    bcall(ess: test1)
=>
    coco

END

&

THEORY test IS

    aaa;
    bunproved(bbb+kkk);
    bunproved(ccc);
    ddd+eee;
    bunproved(fff);
    ggg-fff

END

```

Résultat

```

test1: (2) bbb+kkk
test2: (3) ccc
test3: (5) fff
test4: ECHEC

```

Chapitre 37

blen

`blen(a)`

Paramètres

a : ATOME.

Nature

Opération sur un sous-but.

Utilisation pratique

Pour obtenir la taille d'un atome (en vue de faire des éditions).

Évaluation

Calcule le nombre de caractères de *a*.

Tactique

ARI.

Exemple

```
THEORY ess IS
```

```
  bcall(WRITE: bwritef("%\n",P))
```

```
=>
```

```
  P;
```

```
  bcall((NEWV~;ARI~;ess): (ddd+++blen(blen(bnewv(aaa,bbb,ccc))))+++eee))
```

```
=>
```

```
  coco
```

```
END
```

Résultat

```
ddd+++1+++eee
```


Chapitre 38

blenf

`blenf(f, n)`

Paramètres

f : FORMULE

n : JOKER.

Nature

Garde.

Utilisation pratique

Pour obtenir la taille d'une formule (en nombre de symboles).

Évaluation

Calcule le nombre de symboles de *f* et place le résultat numérique dans *n*.

Exemple

THEORY ess IS

```
blenf((a+b+c),n) &
bguard(WRITE: bwritef("longueur de % : %\n", (a+b+c),n)) &
blenf((ff(uu)),m) &
bguard(WRITE: bwritef("longueur de % : %\n", (ff(uu)),m)) &
blenf(uu,l) &
bguard(WRITE: bwritef("longueur de % : %\n", uu,l)) &
blenf(p,o) &
bguard(WRITE: bwritef("longueur de % : %\n", p,o))
=>
p
```

END

Résultat

longueur de $a+b+c$: 5

longueur de $ff(uu)$: 3

longueur de uu : 1

longueur de $sdfg+asd+asd+asd$: 7

Chapitre 39

blent

`blent(t.n)`

Paramètres

t : NOM_DE_THÉORIE

n : FORMULE

Nature

Garde.

Utilisation pratique

Obtenir le nombre de règles d'une théorie.

Évaluation

La garde est un SUCCES lorsque la théorie *t* existe et que sa taille coïncide avec la formule *n*. Les jokers non instanciés de *n* sont instanciés.

Exemple

```
THEORY ess IS

bbbb;

blent(ess.x) & /* doit etre 4 */
blent(beuh.y) &
blent(aaah.z) &
bcall(WRITE: bwritef("ess: % beuh: % aaah: %\n",x,y,z))
=>
titi;

bcall(ess: titi)
=>
coco

END
```

&

THEORY beuh IS

aaa;aaa;aaa;aaa;aaa;aaa;aaa;aaa;aaa

END

&

THEORY aaah END

Résultat

ess: 4 beuh: 9 aaah: 0

Chapitre 40

blident

`blident(l)`

Paramètres

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_VIRGULES.

Nature

Garde.

Utilisation pratique

Pour tester qu'une liste de formules est une liste d'identificateurs.

Évaluation

SUCCES si la formule *l* est une liste d'IDENTIFICATEURS distincts. On rappelle qu'un IDENTIFICATEUR est une chaîne de caractères de longueur plus grande que 1, et constituée de LETTRES, de CHIFFRES ou du caractère UNDERSCORE.

Exemple

THEORY ess IS

```
bcall(WRITE: bwritef("test2: ECHEC\n"))
```

```
=>
```

```
test3;
```

```
blident(a,bbb,ccc,ddd) &
```

```
bcall(WRITE: bwritef("test2: SUCCES\n"))
```

```
=>
```

```
test3;
```

```
bcall(WRITE: bwritef("test2: ECHEC\n")) &
```

```
test3
```

```
=>
```

```
test2;
```



```
blident(aaa,bbb,ccc,ddd,ddd) &
bcall(WRITE: bwritef("test2: SUCCES\n")) &
test3
=>
test2;

bcall(WRITE: bwritef("test1: ECHEC\n")) &
test2
=>
test1;

blident(aaa,bbb,ccc,ddd_1) &
bcall(WRITE: bwritef("test1: SUCCES\n")) &
test2
=>
test1;

bcall(ess: test1)
=>
coco
```

END

Résultat

```
test1: SUCCES
test2: ECHEC
test3: ECHEC
```

bUpident

Chapitre 41

load

`load(f)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

Pré-conditions

f doit être un nom de fichier, sur lequel on a le droit d'écrire, et sur lequel on a stocké antérieurement des théories en format binaire au moyen de l'opération `bdump`.

Nature

Opération sur un but.

Utilisation pratique

Pour charger le contenu d'un fichier binaire dans des théories.

Évaluation

Il s'agit de l'opération symétrique de l'opération `bdump`. Les théories, préalablement stockées sur le fichier *f* par l'opération `bdump`, sont maintenant lues.

Une règle *r*, stockée sous la forme `bproved(r)`, devient un lemme prouvé. Une règle *r*, stockée sous la forme `bunproved(r)`, devient un lemme non prouvé.

Tactique

RULE.

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul.

Exemple

On suppose que les contenus des fichiers `toto` et `toto.sym` sont ceux correspondant aux résultats de l'exemple de l'opération `bdump`.

THEORY ess IS

```
bcall(RULE: (load("toto"))) &  
bcall((ARI;dmpt)~: (dump(1,aaa) & dump(1,bbb) & dump(2,ccc)))
```

=>

coco

END

&

THEORY dmpt IS

bcall(WRITE: bwritef("END\n\n"))

=>

dumpt(x,n,t.i);

brule(t.i,R) &

bcall(WRITE: bwritef(" %;\n",R)) &

dumpt(x,n,t.(i+1))

=>

dumpt(x,n,t.i);

brule(t.i,R) &

bcall(WRITE: bwritef(" %;\n",bproved(R))) &

dumpt(1,n,t.(i+1))

=>

dumpt(1,n,t.i);

blemma(t.i,R) &

bcall(WRITE: bwritef(" %;\n",bunproved(R))) &

dumpt(x,n,t.(i+1))

=>

dumpt(x,n,t.i);

brule(t.i,R) &

bcall(WRITE: bwritef(" %\n",R)) &

dumpt(x,i,t.(i+1))

=>

dumpt(x,i,t.i);

brule(t.i,R) &

bcall(WRITE: bwritef(" %\n",bproved(R))) &

dumpt(1,i,t.(i+1))

=>

dumpt(1,i,t.i);

blemma(t.i,R) &

bcall(WRITE: bwritef(" %\n",bunproved(R))) &

dumpt(x,i,t.(i+1))

=>

dumpt(x,i,t.i);

```
    blent(t.n) &  
    bcall(WRITE: bwritef("\nTHEORY % IS\n",t)) &  
    dumpt(x,n,t.1)  
=>  
    dump(x,t)
```

END

Résultat

```
THEORY aaa IS  
  bunproved(aaa1)  
END
```

```
THEORY bbb IS  
  bunproved(bbb1);  
  bproved(bbb2)  
END
```

```
THEORY ccc IS  
  ccc1;  
  ccc2;  
  ccc3  
END
```


Chapitre 42

blvar

`blvar(l)`

Paramètres

l : LISTE_DE_VARIABLES

Nature

Garde.

Utilisation pratique

Pour déterminer la liste des variables quantifiées au point de réécriture.

Évaluation

La garde est toujours un SUCCES.

S'il existe au moins une variable quantifiée au point de réécriture, *l* contient la liste de variables quantifiées. Sinon *l* vaut ?.

Par exemple, si le but courant est

`(aa,bb,cc).0<=aa & 0<=bb & 0<=cc => 0<=aa+bb+cc`

et que la règle

```
binhyp(x=0) &  
blvar(Q) &  
x\Q  
=>  
x == 0
```

s'applique, alors *Q* vaudra `(aa,bb,cc)`.

Cette garde est utilisée au sein de règles de réécriture. Elle permet de s'assurer que l'on ne confond pas une variable non quantifiée avec une variable quantifiée. Elle est souvent utilisée en conjonction avec la garde de non liberté `bnfree x \ E`.

`? \ E` est toujours vrai quelque soit *E*.

Exemple

THEORY ess IS

```

bnot(blvar(?))
=>
qq == pp;

!qq.(qq+1>qq)
=>
%ii.(ii: NAT | uu) = oi$5;

blvar(t$i)
=>
(t$i) == ii;

blvar(Q) &
Q\!yx.(yx<yx+oo) &
bcall(WRITE:bwritef("free Q is %\n",Q)) &
%(tt$0).(tt$0: NAT | uu) = oi$5
=>
!yx.(yx<yx+oo);

blvar(Q) &
Q\aa+2) &
bcall(WRITE:bwritef("aa transforme en oo\n"))
=>
aa == oo;

blvar(Q) &
Q\yx+2) &
bcall(WRITE:bwritef("yx transforme en za\n"))
=>
yx == za;

!yx.(yx<yx+aa)
=>
!(zz$0,uu,hh$9).(zz$0+uu+hh$9>0);

blvar(Q) &
bcall(WRITE:bwritef("Q is %\n",Q))
=>
vv == hh;

!(zz$0,uu,vv$9).(zz$0+uu+vv$9>0)
=>
!yy.(yy<yy+1);

blvar(Q) &
bcall(WRITE:bwritef("Q is %\n",Q))

```

=>

xx == yy;

!xx.(xx < xx+1)

=>

coco

END

Résultat

Q is xx

Q is xx

Q is xx

Q is zz\$0,uu,vv\$9

Q is zz\$0,uu,vv\$9

aa transforme en oo

free Q is ?

EXECUTION ABORTED ON GOAL: !pp.(pp+1>pp)

Chapitre 43

bmark

`bmark(t.n)`

Paramètres

t : NOM_DE_THÉORIE

n : NOMBRE

Pré-conditions

t doit être le nom d'une théorie contenant au moins *n* règles.

Nature

Opération sur un but

Utilisation pratique

On utilise cette opération pour rendre un lemme *prouvé*.

Évaluation

La *n^{ème}* règle de la théorie *t* est rendu *prouvé*. S'il s'agissait d'un lemme, ce dernier ne peut plus être accédé par la garde `blemma` qui peut seulement accéder les lemmes *non-prouvés*.

Tactique

MODR

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
blemma(beuh.i,R) &  
bcall(WRITE: bwritef("(%)\n",i,R))  
=>  
tutu;
```

```
    blemma(beuh.i,R) &
    bcall(WRITE: bwritef("(%)\n",i,R)) &
    bcall(MODR: bmark(beuh.i)) &
    tutu
=>
    titi;
```

```
    bcall(ess~:titi)
=>
    coco
```

END

&

THEORY beuh IS

```
    bunproved(cocorico);
    bunproved(Au-revoir);
    bunproved(Bonjour)
```

END

Résultat

- (3) Bonjour
- (2) Au-revoir

Chapitre 44

bmask

bmask

Paramètres

Aucun paramètre.

Nature

Garde.

Utilisation pratique

Retarder l'effet de la réception d'un signal SIGTERM jusqu'au prochain **bunmask**..

Évaluation

L'évaluation de la garde **bmask** est un succès si elle a lieu dans une portion non déjà masquée. De même, l'évaluation de la garde **bunmask** est un succès seulement si elle a lieu dans un portion masquée.

A partir de l'exécution de **bmask**, la réception des signaux SIGTERM est masquée jusqu'au prochain **bunmask**. On définit ainsi une portion masquée pendant laquelle la réception d'un signal SIGTERM est sans effet, elle est simplement mémorisée. Si plusieurs signaux SIGTERM sont reçus dans cette portion masquée, ils sont mémorisés comme une seule interruption. Lors de l'exécution de **bunmask**, le signal mémorisé a l'effet normal d'un SIGTERM : armement de la garde **binter** et échec de l'éventuelle sous preuve lancée par **bguardi**.

Exemple

```
THEORY start IS
```

```
    bcall((toto;ARI~): coco) => p
```

```
END
```

```
&
```

```
THEORY toto IS
```

```
bcall(WRITE: bwritef("bad interrupt memorisation\n"))
=>
coco2;

binter &
bcall(WRITE: bwritef("OK interrupt found\n"))
=>
coco2;

bunmask &
bcall(WRITE: bwritef("bad bunmask eval\n"))
=>
coco2;

bcall(WAIT: bwait(1)) &
coco(x+1)
=>
coco(x);

bmask &
bcall(WRITE: bwritef("bad bmask eval\n"))
=>
coco(x);

bunmask &
bcall(WRITE: bwritef("unmasking...\n")) &
coco2
=>
coco(10);

bmask &
bcall(WRITE: bwritef("masking, counting 10...\n")) &
coco(1)
=>
coco
```

END

Résultat

```
krt -c ess
krt -b ess.kin ess.ex &
[pid = 123]
masking, counting 10...
kill -15 123
(attente...)
unmasking...
OK interrupt found
```

Chapitre 45

bmatch

`bmatch(x, p, q, e)`

Paramètres

x : VARIABLE

p : FORMULE

q : FORMULE

e : FORMULE

Nature

Garde.

Utilisation pratique

Pour vérifier que l'on peut instancier une formule quantifiée.

Évaluation

Pour que la garde soit un SUCCES, il est tout d'abord nécessaire que la formule p ne contienne pas de quantificateurs. Il faut ensuite qu'il existe une formule f telle que le remplacement de x par f dans p soit identique à la formule q . Il faut, enfin, que cette formule f coïncide avec e , supposée non instanciée (la plupart du temps, e est un simple joker). Les jokers non instanciés de e sont instanciés.

Exemple

```
THEORY ess IS
```

```
    binhyp(!x.(H=>P)) &
    bmatch(x,P,Q,E) &
    bcall((SUB;WRITE):
    bwritef("P: %\nx: %\nE: %\n[x:=E]P: %\nQ:          %\n",P,x,E,[x:=E]P,Q))
=>
Q;

( !(xx$1,yy).(qq(xx$1,yy) => pp(xx$1,ff(xx$1),yy))
=>
```

```
    pp(aa,ff(aa),bb)
  )
=>
  ccc;

  bcall((ess;DED;ess):ccc)
=>
  coco
```

END

Résultat

```
P: pp(xx$1,ff(xx$1),yy)
x: xx$1,yy
E: aa,bb
[x:=E]P: pp(aa,ff(aa),bb)
Q:      pp(aa,ff(aa),bb)
```

Chapitre 46

bmodr

$\text{bmodr}(t.n, f)$

Paramètres

t : NOM_DE_THÉORIE

n : NOMBRE

f : FORMULE

Pré-conditions

t doit être le nom d'une théorie contenant au moins n règles.

Nature

Opération sur un but

Utilisation pratique

On utilise cette opération pour modifier une règle déjà créée.

Évaluation

La $n^{\text{ème}}$ règle de la théorie t est remplacée par f .

Tactique

MODR

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
brule(beuh.i,R) &  
bcall(WRITE: bwritef("% \n",R))  
=>  
tutu(i);
```



```
brule(beuh.i,R) &
bcall(WRITE: bwritef("% \n",R)) &
bcall(MODR: bmodr(beuh.i, Au-Revoir)) &
tutu(i)
=>
titi(i);

bcall(ess~:titi(3))
=>
coco
```

END

&

THEORY beuh IS

```
cocorico;
aaaah;
Bonjour
```

END

Résultat

```
Bonjour
Au-Revoir
```

Chapitre 47

bnewv

`bnewv(l)`

Paramètres

l : LISTE_D_IDENTIFICATEURS_DE_VARIABLES.

Nature

Opération sur un sous-but.

Utilisation pratique

Pour construire de nouveaux identificateurs de variables.

Évaluation

Les différents constituants de la liste sont concaténés.

Tactique

NEWV.

Exemple

```
THEORY ess IS
```

```
  bcall(WRITE: bwritef("%\n",P))
```

```
=>
```

```
  P;
```

```
  bcall((NEWV~;ess): (ddd+++bnewv(aaa,bnewv(bbb,ccc))+++eee))
```

```
=>
```

```
  coco
```

```
END
```

Résultat

```
ddd+++aaabbbccc+++eee
```


Chapitre 48

bnfree

$v \setminus f$

Paramètres

v : VARIABLE

f : FORMULE

Nature

Garde.

Utilisation pratique

Pour savoir si une variable n'est pas libre dans une formule.

Évaluation

La garde est un SUCCES si la VARIABLE v n'a pas d'occurrences libres dans f

Exemple

```
THEORY ess IS
```

```
  bnot(xx \ xx+45) &  
  bcall(WRITE: bwritef("ECHEC\n"))  
=>
```

```
  titi;
```

```
  xx \ yy+45 &  
  bcall(WRITE: bwritef("SUCCES\n")) &  
  titi
```

```
=>
```

```
  coco
```

```
END
```

Résultat

```
SUCCES
```

ECHEC

Chapitre 49

bnlmap

$$g \text{ bnlmap } l \qquad f(e_1, \dots, e_n) \text{ bnmap } l$$

Paramètres

g : FORMULE_NON_FONCTIONELLE

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_V_PV_A

f : FORMULE

e_i : FORMULE

N.B. : Une formule de la forme $h(e_1, \dots, e_n)$ où h est un OPÉRATEUR_UNAIRE est une FORMULE_NON_FONCTIONELLE

Nature

Opération sur un sous-but

Utilisation pratique

On utilise cette opération pour distribuer une formule sur une liste.

Évaluation

On suppose que la liste l est de la forme : $l_1 \text{ OP } \dots \text{ OP } l_m$. On rappelle que OP est soit une VIRGULE, soit un POINT-VIRGULE, soit enfin un AMPERSAND. L'évaluation de l'opération **bnmap** dépend de sa forme particulière :

- Pour la première forme, l'évaluation consiste à générer le sous-but suivant :
 $g(l_1, 1, m) \text{ OP } \dots \text{ OP } g(l_m, m, m)$.
- Pour la deuxième forme, l'évaluation consiste à générer le sous-but suivant :
 $f(e_1, \dots, e_n, l_1, 1, m) \text{ OP } \dots \text{ OP } f(e_1, \dots, e_n, l_m, m, m)$

L'évaluation de cette opération est donc pratiquement la même que celle de l'opération **bslmap**. On ajoute seulement, à chaque élément distribué, l'index de l'élément correspondant dans la liste initiale, de même que la taille de la dite liste.

Tactique

LMAP

Exemple

THEORY ess IS

```

    bcall(WRITE: bwritef("%\n",P))
=>
    P;

    bcall((LMAP;ess)~: beuh(tata bnlmap (a;b;c))) &
    bcall((LMAP;ess)~: beuh(bwritef(aaa) bnlmap (a&b&c))) &
    bcall((LMAP;ess)~: beuh(titi(aaa) bnlmap (a,b,c)))
=>
    coco

```

END

Résultat

```

beuh(tata(a,1,3);tata(b,2,3);tata(c,3,3))
beuh(bwritef(aaa)(a,1,3) & bwritef(aaa)(b,2,3) & bwritef(aaa)(c,3,3))
beuh(titi(aaa,a,1,3),titi(aaa,b,2,3),titi(aaa,c,3,3))

```

Chapitre 50

bnmap

$$g \text{ bnmap } l \qquad f(e_1, \dots, e_n) \text{ bnmap } l$$

Paramètres

g : FORMULE_NON_FONCTIONELLE

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_V_PV_A

f : FORMULE

e_i : FORMULE

N.B. : Une formule de la forme $h(e_1, \dots, e_n)$ où h est un OPÉRATEUR_UNAIRE est une FORMULE_NON_FONCTIONELLE

Pré-conditions

Néant

Nature

Opération sur un but

Utilisation pratique

On utilise cette opération pour distribuer un but, opérant sur une liste, en autant de buts distincts opérant sur chacun des éléments de la liste.

Évaluation

On suppose que la liste l est de la forme : $l_1 \text{ OP } \dots \text{ OP } l_m$. On rappelle que OP est soit une VIRGULE, soit un POINT-VIRGULE, soit enfin un AMPERSAND. L'évaluation de l'opération **bnmap** dépend de sa forme particulière :

- Pour la première forme, l'évaluation consiste à générer les m buts suivants : $g(l_1, 1, m,), \dots, g(l_m, m, m,)$.
- Pour la deuxième forme, l'évaluation consiste à générer les m buts suivants : $f(e_1, \dots, e_n, l_1, 1, m,), \dots, f(e_1, \dots, e_n, l_m, m, m,)$

L'évaluation de cette opération est donc pratiquement la même que celle de l'opération **bmap**. On ajoute seulement, à chaque nouveau but créé, l'index de l'élément correspondant dans la liste initiale, de même que la taille de la dite liste.

Tactique

MAP

Exemple

THEORY ess IS

```

    bcall(WRITE: bwritef("%\n",P))
=>
    P;

    bcall((MAP;ess)~: (titi bnmap (4,5,6))) &
    bcall(WRITE: bwritef("\n")) &
    bcall((MAP;ess)~: (bwritef(aaa) bnmap (4;5;6))) &
    bcall(WRITE: bwritef("\n")) &
    bcall((MAP;ess)~: (tutu(aaa,bbb) bnmap (4&5&6)))
=>
    coco

```

END

Résultat

```

titi(4,1,3)
titi(5,2,3)
titi(6,3,3)

bwritef(aaa)(4,1,3)
bwritef(aaa)(5,2,3)
bwritef(aaa)(6,3,3)

tutu(aaa,bbb,4,1,3)
tutu(aaa,bbb,5,2,3)
tutu(aaa,bbb,6,3,3)

```

Chapitre 51

bnot

`bnot(g)`

Paramètres

g : GARDE

Nature

Garde.

Utilisation pratique

Pour simplifier l'usage des gardes en échec.

Évaluation

La garde `bnot` est un SUCCES, si la garde *g* est un ÉCHEC.

Exemple

```
THEORY ess IS
```

```
  bcall(WRITE: bwritef("Salut \n"))
```

```
=>
```

```
  aaa(n);
```

```
  breade("Ecrire >> (en instanciant les ? par des nombres): ",a>b) &  
  bnot(btest(a>b)) &
```

```
  bcall(WRITE: bwritef("Hello \n")) &  
  aaa(n+1)
```

```
=>
```

```
  aaa(n)/*(a,b)*/;
```

```
  bcall((ARI;ess)~: aaa(1))
```

```
=>
```

```
  coco
```

```
END
```

Résultat

```
Ecrire >>? (en instanciant les ? par des nombres): 3>5  
Hello  
Ecrire >>? (en instanciant les ? par des nombres): 4>9  
Hello  
Ecrire >>? (en instanciant les ? par des nombres): 5>1  
Salut
```

bnum

```
    bcall(WRITE: bwritef("test2: SUCCES\n")) &
    test3
=>
    test2;

    bcall(WRITE: bwritef("test1: ECHEC\n")) &
    test2
=>
    test1;

    bnum(35) &
    bcall(WRITE: bwritef("test1: SUCCES\n")) &
    test2
=>
    test1;

    bcall(ess: test1)
=>
    coco
```

END

Résultat

```
test1: SUCCES
test2: ECHEC
test3: ECHEC
```

Chapitre 53

bpattern

$\text{bpattern}(f, g)$

Paramètres

f : FORMULE

g : FORMULE

Nature

Garde.

Utilisation pratique

Savoir si une formule coïncide avec une autre.

Évaluation

La garde est un SUCCES si la formule f coïncide avec la formule g , supposée non instanciée. Les jokers non instanciés de g sont instanciés.

Exemple

```
THEORY ess IS

  bnot(bpattern(p,q)) &
  bcall(WRITE: bwritef("ECHEC"))
=>
  titi(p,q);

  bpattern(aaa+bbb,a+b) &
  bcall(WRITE: bwritef("%    %\n",a,b)) &
  titi(aaa+bbb,k+1)
=>
  coco

END
```

Résultat

aaa bbb
ECHEC

Chapitre 54

bpop

`bpop(t)`

Paramètres

t : NOM_DE_THÉORIE

Pré-conditions

- La théorie *t* doit exister
- La dernière règle de la théorie *t* doit avoir été ajoutée en cours de preuve par l'une des opérations `bcrer`, `bcrel` ou `bcrelr`.

Nature

Opération sur un but

Utilisation pratique

On utilise cette opération lorsque l'on se sert d'une théorie comme d'une pile.

Évaluation

Supprimer la dernière règle de la théorie *t*.

Tactique

RULE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

```
THEORY ess IS

  blent(th2.i) &
  bcall(WRITE:bwritef("len3: %\n",i))
=>
  tutu;

  blent(th2.i) &
```



```

    bcall(WRITE:bwritef("len2: %\n",i)) &
    bcall(RULE: bpop(th2)) &
    tutu
=>
    tata;

    blent(th2.i) &
    bcall(RULE: bcrelr(th2,fff)) &
    bcall(WRITE:bwritef("len1: %\n",i)) &
    tata
=>
    titi;

    bcall(RULE:bpop(th1) | WRITE : bwritef("th1 n'existe pas\n")) &
    bcall(RULE:bpop(th2) | WRITE : bwritef("th2 n'a pas de regles nouvelles\n")) &
    titi
=>
    tete;

    bcall(ess:tete)
=>
    coco

```

END

&

THEORY th2 IS

```

    aaa;
    bbb;
    ccc;
    ddd;
    eee

```

END

Résultat

```

th1 n'existe pas
th2 n'a pas de regles nouvelles
len1: 5
len2: 6
len3: 5

```

Chapitre 55

bprintf

`bprintf(f)` `bprintf(f, l)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_VIRGULES

Nature

Opération sur un but

Utilisation pratique

Écriture formatée sur un fichier texte

Évaluation

Chaque caractère de *f* (éventuellement ré-interprété) est écrit sur un certain fichier.

- **Détermination du fichier** : Il s'agit du fichier mentionné dans la dernière garde `bconnect` ou `bappend` exécutée avec succès, à moins que l'opération `bclose` n'ait été exécutée entre temps. Par défaut, il s'agit du fichier de sortie standard.
- **Interprétation des caractères** : Les caractères sont normalement interprétés littéralement. Seuls les caractères ou groupes de caractères qui suivent sont interprétés de façon particulière :
 - `\B` est la sonette
 - `\E` est l'échappement
 - `\t` est la tabulation
 - `\n` est le saut de ligne
 - `\"` est `"`
 - `\\` est `\`
 - `\%` est `%`
 - `%`, non précédé de `\` et non suivi d'une chaîne numérique. L'interprétation est différente suivant la nature de l'opération :
 - `bprintf(f)` : aucune impression n'est effectuée
 - `bprintf(f, l)` : la $n^{\text{ème}}$ occurrence de `%` est remplacée par la formule de rang $((n - 1) \bmod t)$ de *l* (*t* est la longueur de la liste *l*).

- %, non précédé de \ et suivi d'une chaîne numérique désignant un nombre m . L'interprétation est la même que dans le cas précédent sauf lorsque la formule à imprimer est un NOMBRE ou un IDENTIFICATEUR. Dans chacun de ces deux cas, on imprime au moins m caractères ; pour cela, lorsque la formule à imprimer ne contient pas assez de caractères, on insère, en tête de la dite formule, autant de caractères blancs qu'il est nécessaire pour obtenir exactement le nombre de caractères requis.

Tactique

WRITE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
bconnect("toto") &
bcall(WRITE: bprintf("\BA%A%2A\t345\taaa\t\" \\ \%tbbb\nBBB\n\n")) &
bcall(WRITE: bprintf("plus1: % +++ plus2: %\n\n",a+b)) &
bcall(WRITE: bprintf("%7 ^^^ %4 ^^^ %1 ^^^ %2 ^^^\n\n",250,aaa)) &
bcall(SHELL: bshell("cat toto"))
=>
coco
```

END

Résultat

```
AAA 345 aaa " \ % bbb
BBB
```

```
plus1: a+b +++ plus2: a+b
```

```
250 ^^^ aaa ^^^ 250 ^^^ aaa ^^^
```

Chapitre 56

breade

`breade(f, (l), g) breade(f, g) breade(g)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

l : LISTE_DE_FORMULES_SÉPARÉS_PAR_DES_VIRGULES

g : FORMULE

Nature

Garde.

Utilisation pratique

Rentrer une information depuis le terminal.

Évaluation

Lorsque la garde a la forme `breade(f, (l), g)` ou la forme `breade(f, g)`, son évaluation est précédée de l'exécution de l'opération `bwritef(f, l)` ou `bwritef(f)`.

On suppose ensuite que l'utilisateur rentre une certaine formule *h*, limitée par le “retour-charriot”. Si l'utilisateur frappe le “retour-charriot” immédiatement, on considère qu'il a rentré, l'IDENTIFICATEUR **CR**.

La garde est un **SUCCES** lorsque la formule *h* coïncide avec la formule *g*. Les **JOKERS** non instanciés de *g* sont instanciés.

Exemple

```
THEORY ess IS
```

```
    breade("%: ",Hello,t) &
    bcall(WRITE: bwritef("echo:  %\n",t))
=>
    titi;

    bcall(ess: titi)
=>
```

coco

END

Résultat

Hello: cocorico
echo: cocorico

bwritef

Chapitre 57

breadf

`breadf(f, (l), g)` `breadf(f, g)` `breadf(g)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

l : LISTE_DE_FORMULES_SÉPARÉS_PAR_DES_VIRGULES

g : FORMULE

Nature

Garde.

Utilisation pratique

Rentrer une information depuis le terminal.

Évaluation

Lorsque la garde a la forme `breadf(f, (l), g)` ou la forme `breadf(f, g)`, son évaluation est précédée de l'exécution de l'opération `bwritef(f, l)` ou `bwritef(f)`.

On suppose ensuite que l'utilisateur rentre une certaine formule *h*, limitée par le symbole ANTIQUOTE. On peut ainsi rentrer des formules sur plusieurs lignes. La garde est un SUCCES lorsque la formule *h* coïncide avec la formule *g*. Les JOKERS non instanciés de *g* sont instanciés.

Exemple

```
THEORY ess IS
```

```
breadf("": ",Hello,t) &
bcall(WRITE: bwritef("echo:  %\n",t))
=>
titi;

bcall(ess: titi)
=>
coco
```

END

Résultat

```
Hello: cocorico++  
cicirici'  
echo: cocorico++cicirici
```

indexbreade

Chapitre 58

brecompact

brecompact

Nature

Opération sur un but.

Utilisation pratique

Pour recompacter la mémoire dynamique.

Évaluation

Recompacte la mémoire dynamique avec des espaces libres de 20 mots au maximum.

Chapitre 59

bresetcomp

bresetcomp

Nature

Opération sur un but.

Utilisation pratique

Pour nettoyer l'espace de compilation dynamique.

Évaluation

L'espace de compilation est nettoyé.

Tactique

RULE.

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul.

Exemple

THEORY ess IS

```
bget("theo2",R) &  
bcrelr(aaa,R) &  
bcompile(aaa) &  
bcall(aaa~: tutu2)
```

=>

```
beuh;
```

```
bget("theo1",R) &  
bcrelr(aaa,R) &  
bcompile(aaa) &  
bcall(aaa~: tutu1) &  
bclean(aaa) &  
bresetcomp &
```

```

    beuh
=>
    titi;

    bcall((RULE;ess)~: titi)
=>
    coco

```

END

&

THEORY aaa END

&

THEORY bbb END

Fichier Theo1

```

    bcall(WRITE:bwritef("Hello_1\n")) &
    toto
=>
    tutu1;

    bcall(WRITE:bwritef("Hello_2\n"))
=>
    toto

```

Fichier Theo2

```

    bcall(WRITE:bwritef("beuh_1\n")) &
    toto2
=>
    tutu2;

    bcall(WRITE:bwritef("beuh_2\n")) &
    tata2
=>
    toto2;

    bcall(WRITE:bwritef("beuh_3\n"))
=>
    tata2

```

Résultat

```

Hello_1
Hello_2

```

beuh_1

beuh_2

beuh_3

Chapitre 60

bresult

`bresult(f)`

Paramètres

f : FORMULE

Pré-conditions

On doit se trouver sous le contrôle d'une garde `bguard`.

Nature

Opération sur un but.

Utilisation pratique

Pour stocker le résultat de l'exécution d'une `bguard`.

Tactique

RES

Évaluation

La FORMULE *f* est stockée. Elle pourra être consultée plus tard par la garde `bgetresult`.

Exemple

```
THEORY ess IS

  bguard(tst: z,r) &
  bcall(WRITE: bwritef("res: %\n",r))
=>
  bresult(z);

  bresult(aaa)
=>
  beuh;

  bcall((ess;RES): beuh)
=>
```

coco

END

&

THEORY tst IS

bcall(RES: bresult(bbb))

=>

aaa

END

Résultat

res: bbb

Chapitre 61

brev

`brev(l)`

Paramètres

l : LISTE_ DE_FORMULES_SÉPARÉES_PAR_DES_V_PV_A

Nature

Opération sur un sous-but.

Utilisation pratique

Renversement d'une liste.

Évaluation

Renverse la liste *l*.

Tactique

REV.

Exemple

```
THEORY ess IS
```

```
  bcall(WRITE: bwritef("%\n",P))
```

```
=>
```

```
  P;
```

```
  bcall((REV~;ess): (brev(aaa,brev(eee+ccc),brev(ff1;ff2),brev(hhh&kkk))))
```

```
=>
```

```
  coco
```

```
END
```

Résultat

```
(kkk & hhh),(ff2;ff1),eee+ccc,aaa
```


Chapitre 62

brule

`brule(t.n, f)`

`brule(m, t.n, f)`

Paramètres

t : NOM_DE_THEORIE

n : FORMULE

f : FORMULE

m : NOMBRE

Nature

Garde.

Utilisation pratique

Rechercher une règle dans une théorie.

Évaluation

Nous devons considérer différents cas suivant la nature de *n* :

- Lorsque *n* est un NOMBRE, la garde est un SUCCES si la règle *n* de la théorie *t* existe et coïncide avec la FORMULE *f*. Les JOKERS non instanciés de *f* sont instanciés.
- Lorsque *n* est un JOKER, la garde est un SUCCES s’il existe une règle de la théorie *t* qui coïncide avec la FORMULE *f*. Lorsqu’il y a plusieurs règles qui coïncident avec *f*, on choisit la dernière d’entre elles. Le JOKER *n* est alors instancié avec le numéro de la règle sélectionnée. Les JOKERS non instanciés de *f* sont instanciés.
- Dans tous les autres cas, la garde est un ECHEC.

L’évaluation de la garde `brule(m, t.n, f)` correspond à ce que nous venons d’expliquer, à ceci près que, dans le cas où *n* est un JOKER, la règle sélectionnée est la dernière règle, qui coïncide avec *f*, et dont le numéro est inférieur ou égal à *m*.

Exemple

```
THEORY ess IS
```

```
bcall(WRITE: bwritef("test4: ECHEC\n"))
```

```

=>
  test4;

  brule(test.j,r-s) &
  bcall(WRITE: bwritef("test4: (%) % \n",j,r-s))
=>
  test4;

  brule(test.5,r) &
  bcall(WRITE: bwritef("test3: (5) %\n",r)) &
  test4
=>
  test3;

  brule(3,test.j,r) &
  bcall(WRITE: bwritef("test2: (%) %\n",j,r)) &
  test3
=>
  test2;

  brule(test.j,r+s) &
  bcall(WRITE: bwritef("test1: (%) % \n",j,r+s)) &
  test2
=>
  test1;

  bcall(ess: test1)
=>
  coco

END
&

THEORY test IS

  aaa;bbb+kkk;ccc;ddd+eee;fff;ggg

END

```

Résultat

```

test1: (4) ddd+eee
test2: (3) ccc
test3: (5) fff
test4: ECHEC

```

Chapitre 63

bsearch

$\text{bsearch}(p, l, r)$ $\text{bsearch}(p, l, r, s)$

Paramètres

p : FORMULE

l : FORMULE_NON_ATOMIQUE

r : FORMULE

s : FORMULE

Nature

Garde.

Utilisation pratique

Pour rechercher, et éventuellement modifier, un élément dans une liste.

Évaluation

La formule l est de la forme $l_1 \text{ op } \dots \text{ op } l_i \text{ op } \dots \text{ op } l_n$ où n supérieur ou égal à 2 et où op est un OPÉRATEUR_BINAIRE. On doit considérer deux cas suivant la forme de la garde :

- La garde $\text{bsearch}(p, l, r)$ est un SUCCES lorsqu'il existe une sous formule l_i qui coïncide avec p et lorsque la formule, que l'on obtient en enlevant de l la sous-formule l_i , coïncide avec la formule r .
- La garde $\text{bsearch}(p, l, r, s)$ est un SUCCES lorsqu'il existe une sous formule l_i de l qui coïncide avec p , et lorsque la formule, que l'on obtient en enlevant de l la sous-formule l_i et en la remplaçant par la formule s dûment instanciée, coïncide avec la formule r .

Les jokers non instanciés de p , r et s sont instanciés.

Exemple

THEORY ess IS

```
bsearch((a-{x}),(aaa \/ (bbb-{xx}) \/ ccc \/ (ddd \/ {xx}) \/ eee),r,a) &
bsearch((b\/{x}),r,s,b) &
bsearch((b\/{x}),(aaa \/ (bbb-{xx}) \/ ccc \/ (ddd \/ {xx}) \/ eee),h) &
```

```
bcall(WRITE: bwritef("r: %\na: %\nb: %\ns: %\nh: %\n",r,a,b,s,h))
=>
coco
```

END

Résultat

```
r: aaa\bbb\ccc\(ddd\{xx})\eee
a: bbb
b: ddd
s: aaa\bbb\ccc\ddd\eee
h: aaa\bbb-{xx}\ccc\eee
```

Chapitre 64

bsetmode

bsetmode(*n*)

Paramètres

n : NOMBRE

Nature

Opération sur un but

Utilisation pratique

Pour changer le mode de preuve, qui définit certaines caractéristiques modifiables du **Logic-Solver** comme le fait de décharger systématiquement toute égalité triviale.

Évaluation

L'évaluation de l'opération **bsetmode(*n*)** consiste à mettre le **Logic-Solver** dans le mode de preuve *n*. Le **Logic-Solver** interprète *n* de la manière suivante :

- décomposition en base 2 de *n* :
 $n = x_1 2^0 + x_2 2^1 + \dots + x_p 2^{p-1}$ avec $x_i = 0$ ou 1
chaque x_i correspond alors à une option facultative, activée si $x_i = 1$.

Les options facultatives actuellement définies sont :

- x_1 : empêche l'application des règles implicites **EQUAL**, **INHYP** et **FALSE** qui normalement éliminent tout but qui est de la forme $a = a$ ou qui est en hypothèse, ou encore dont la pile d'hypothèses contient un **bfalse**.
- x_2 : provoque l'impression d'un message d'erreur :
"stopping forward because more than ... hypothesis generated"
si une tactique par l'avant est arrêtée parceque le nombre maximal d'hypothèses générables par une même tactique avant est atteint (paramètre réglable, option **-a dxxx** de **krt**). Attention : x_2 ne concerne que le message, l'arrêt de la tactique avant sur maximum atteint se produit même si x_2 n'est pas actif.

La valeur des x_i pour $i > 2$ est sans influence.

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```

    bsetmode(toto) &
    bcall(pr: (toto = toto)) &
    bcall((DED;pr): (bfalse => BfalseNonElim)) &
    bsetmode(1) &
    bcall(pr: (toto = toto)) &
    bcall((DED;pr): (bfalse => BfalseNonElim)) &
    bcall((DED;pr): (hh => hh)) &
    bcall((DED;pr),fwd: (hh => glop1)) &
    bsetmode(2) &
    bcall((DED;pr),fwd: (hh => glop1))
=>
P

```

END

&

THEORY pr IS

```

    bcall(WRITE: bprintf("%\n", P))
=>
P

```

END

&

THEORY fwd IS

```

    h
=>
    h+hh

```

END

Résultat

```

krt: ignored instruction bsetmode(toto)
toto = toto
BfalseNonElim
hh

```

SEQUENCE MEMORY OVERFLOW : STOP

Chapitre 65

bshell

`bshell(f)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

Pré-conditions

La chaîne *f* doit pouvoir être évaluée avec succès par le SHELL de l'installation.

Nature

Opération sur un but

Utilisation pratique

Cette opération est conservée par compatibilité avec les versions antérieures. La fonctionnalité à employer pour lancer un programme extérieur est `bfork`.

Évaluation

Le contrôle est donné au SHELL qui interprète la chaîne *f*.

Tactique

SHELL

Recul

L'effet de bord du à cette opération n'est évidemment pas détruit lors d'un recul

Exemple

```
THEORY ess IS
```

```
    bcall(SHELL: bshell("ls;echo Bonjour") | WRITE : bwritef("Mauvais shell\n")) &  
    bcall(SHELL: bshell("beuh") | WRITE : bwritef("Mauvais shell\n"))  
=>  
    coco
```

```
END
```

Résultat


```
ess      ess.ex  ess.kin  tctfile  thtfile
Bonjour
sh: beuh: not found
Mauvais shell
```

Chapitre 66

bslmap

$$g \text{ bslmap } l \qquad f(e_1, \dots, e_n) \text{ bslmap } l$$

Paramètres

g : FORMULE_NON_FONCTIONELLE

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_V_PV_A

f : FORMULE

e_i : FORMULE

N.B. : Une formule de la forme $h(e_1, \dots, e_n)$ où h est un OPÉRATEUR_UNAIRE est une FORMULE_NON_FONCTIONELLE

Nature

Opération sur un sous-but

Utilisation pratique

On utilise cette opération pour distribuer une sous-formule sur une liste.

Évaluation

On suppose que la liste l est de la forme : $l_1 \text{ OP } \dots \text{ OP } l_m$. On rappelle que OP est soit une VIRGULE, soit un POINT-VIRGULE, soit enfin un AMPERSAND. L'évaluation de l'opération **bslmap** dépend de sa forme particulière :

- Pour la première forme, l'évaluation consiste à générer le sous-but suivant :
 $g(l_1) \text{ OP } \dots \text{ OP } g(l_m)$.
- Pour la deuxième forme, l'évaluation consiste à générer le sous-but suivant :
 $f(e_1, \dots, e_n, l_1) \text{ OP } \dots \text{ OP } f(e_1, \dots, e_n, l_m)$

Tactique

LMAP

Exemple

THEORY ess IS

```
bcall(WRITE: bwritef("%\n",P))
```

```
=>
  P;

  bcall((LMAP;ess)~: beuh(tata bslmap (4,5,6))) &
  bcall((LMAP;ess)~: beuh(titi(aaa) bslmap (4,5,6)))
=>
  coco
```

END

Résultat

```
beuh(tata(4),tata(5),tata(6))
beuh(titi(aaa,4),titi(aaa,5),titi(aaa,6))
```

Chapitre 67

bsmap

$$g \text{ bsmap } l \qquad f(e_1, \dots, e_n) \text{ bsmap } l$$

Paramètres

g : FORMULE_NON_FONCTIONELLE

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_V_PV_A

f : FORMULE

e_i : FORMULE

N.B. : Une formule de la forme $h(e_1, \dots, e_n)$ où h est un OPÉRATEUR_UNAIRE est une FORMULE_NON_FONCTIONELLE

Pré-conditions

Néant

Nature

Opération sur un but

Utilisation pratique

On utilise cette opération pour distribuer un but, opérant sur une liste, en autant de buts distincts opérant sur chacun des éléments de la liste.

Évaluation

On suppose que la liste l est de la forme : $l_1 \text{ OP } \dots \text{ OP } l_m$. On rappelle que OP est soit une VIRGULE, soit un POINT-VIRGULE, soit enfin un AMPERSAND. L'évaluation de l'opération **bsmap** dépend de sa forme particulière :

- Pour la première forme, l'évaluation consiste à générer les m buts suivants :
 $g(l_1), \dots, g(l_m)$.
- Pour la deuxième forme, l'évaluation consiste à générer les m buts suivants :
 $f(e_1, \dots, e_n, l_1), \dots, f(e_1, \dots, e_n, l_m)$

Tactique

MAP

Exemple

THEORY ess IS

```
    bcall(WRITE: bwritef("%\n",P))
=>
    P;

    bcall((MAP;ess)~: (titi bsmap (4,5,6))) &
    bcall(WRITE: bwritef("\n")) &
    bcall((MAP;ess)~: (bwritef(aaa) bsmap (4;5;6))) &
    bcall(WRITE: bwritef("\n")) &
    bcall((MAP;ess)~: (toto(aaa,bbb) bsmap (4&5&6)))
=>
    coco
```

END

Résultat

```
titi(4)
titi(5)
titi(6)

bwritef(aaa)(4)
bwritef(aaa)(5)
bwritef(aaa)(6)

toto(aaa,bbb,4)
toto(aaa,bbb,5)
toto(aaa,bbb,6)
```

Chapitre 68

bsparemem

`bsparemem(n)`

Paramètres

n : JOKER

Nature

Garde.

Utilisation pratique

Pour déterminer le pourcentage de mémoire dynamique libre.

Évaluation

Le JOKER *x* est instancié avec le pourcentage (compris donc entre 0 et 100) de mémoire dynamique libre.

Exemple

```
THEORY ess IS

    bsparemem(x) &
    bcall(WRITE: bwritef("res: %\n",x))
=>
coco

END
```

Résultat

```
res: 99
```


Chapitre 69

bstatistics

bstatistics

Paramètres

Aucun paramètre.

Nature

Opération sur un but.

Utilisation pratique

Pour afficher les informations internes du kernel.

Évaluation

Cette opération affiche certaines informations d'encombrement mémoire relatives au kernel :

- nombre d'entiers
- nombre de caractères
- pile des buts
- pile des hypothèses
- nombre de théories
- tableau de compilation (nombre d'instructions générées)

Exemple

```
THEORY ess IS
```

```
    bstatistics  
=>  
P
```

```
END
```

Résultat

```
Integer Sequences: Max: 200000 Created: 27 Free: 0 Used: 27,  
                  Free memory: 3999973 Total memory: 4000000
```


Character Sequences: Max: 10000 Created: 143 Free: 0 Used: 143,
Free memory: 499270 Total memory: 500000
Goal Stacks: Size: 2x10000 Proof branch size: 3 Pending goals: 0
Hypothesis Stack: Size: 10000 Nb of hyps: 0
Theories: Size: 800 Nb of theories: 1
Compiler: Size: 2x50000 Theories: 5 Tactics: 5

Chapitre 70

bstring

`bstring(f)`

Paramètres

f : FORMULE.

Nature

Garde.

Utilisation pratique

Pour tester qu'une formule est une CHAINE_ENTRE_GUILLEMETS.

Évaluation

SUCCES si la formule *f* est une CHAINE_ENTRE_GUILLEMETS. On rappelle qu'une CHAINE_ENTRE_GUILLEMETS est une suite de caractères commençant par un GUILLEMET et se terminant par un GUILLEMET. On peut trouver un GUILLEMET dans la chaîne à condition qu'il soit précédé du caractère l'ANTI_SLASH.

Exemple

THEORY ess IS

```
bcall(WRITE: bwritef("test3: ECHEC\n"))
```

```
=>
```

```
test3;
```

```
bstring(aa+bb) &
```

```
bcall(WRITE: bwritef("test3: SUCCES\n"))
```

```
=>
```

```
test3;
```

```
bcall(WRITE: bwritef("test2: ECHEC\n")) &
```

```
test3
```

```
=>
```

```
test2;
```

```
    bstring(aaa) &
    bcall(WRITE: bwritef("test2: SUCCES\n")) &
    test3
=>
    test2;

    bcall(WRITE: bwritef("test1: ECHEC\n")) &
    test2
=>
    test1;

    bstring("Bonjour \"Monsieur\"") &
    bcall(WRITE: bwritef("test1: SUCCES\n")) &
    test2
=>
    test1;

    bcall(ess: test1)
=>
    coco

END
```

Résultat

```
test1: SUCCES
test2: ECHEC
test3: ECHEC
```

Chapitre 71

bsubfrm

$\text{bsubfrm}(g, d, p, q) \quad \text{bsubfrm}(g, d, p, (q, v))$

Paramètres

g : FORMULE

d : FORMULE

p : FORMULE

q : FORMULE

v : FORMULE

Nature

Garde.

Utilisation pratique

Tester la présence d'une sous-formule dans une formule et la remplacer par une autre.

Évaluation

Pour que la garde soit un SUCCES, il est d'abord nécessaire qu'il existe une sous-formule f de p qui coïncide avec g (supposée non instanciée). On considère ensuite la formule p' , obtenue en remplaçant, dans p , la sous-formule f par d , dûment instanciée. Il faut que cette formule p' coïncide avec q , supposée non instanciée (la plupart du temps, q est un simple joker).

Dans le cas de la deuxième forme de la garde, on considère enfin la liste des variables quantifiées dont dépend la sous-formule f . S'il n'y a pas de telles variables, on considère, par convention, la liste formée du seul symbole ?. Il faut que cette liste coïncide avec v , supposée non instanciée (la plupart du temps, v est un simple joker).

Les jokers non instanciés de g , q et, éventuellement, v sont instanciés.

Exemple

THEORY ess IS

$\text{bsubfrm}(x/:s, \text{not}(x:s), \#(y,z).!(xxx\$1,x,bbb).(x/:s \Rightarrow aaa), (q,v)) \ \&$

```
    bcall((SUB;WRITE): bwritef("q: %\nv: %\n",q,v))  
=>  
    coco
```

```
END
```

Résultat

```
q: #(y,z).!(xxx$1,x,bbb).(not(x: s) => aaa)  
v: xxx$1,x,bbb,y,z
```

Chapitre 72

btac

$\text{btac}(t, f)$

Paramètres

t : NOM_DE_THÉORIE

f : FORMULE

Nature

Opération sur un but.

Utilisation pratique

Pour créer une théorie.

Évaluation

Le deuxième paramètre f ne sert à rien. L'effet de cette opération est de créer la théorie t , si elle ne l'était déjà.

Tactique

TACTIC.

Recul

L'effet de bord de cette opération n'est pas détruit lors d'un recul.

Exemple

```
THEORY ess IS
```

```
    blent(aaa.n) &
    bcall(WRITE: bwritef("Taille de aaa: %\n",n))
=>
    toto;

    btac(aaa,1) &
    bcall(WRITE: bwritef("Creation de aaa\n")) &
    toto
=>
```

```
titi;

blent(aaa.n) &
bcall(WRITE: bwritef("taille de aaa: %\n",n))
=>
titi;

bcall((DED;TACTIC;ess)~: titi)
=>
coco

END
```

Résultat

```
Creation de aaa
Taille de aaa: 0
```

Chapitre 73

btest

`btest(m op n)`

Paramètres

m : FORMULE

n : FORMULE

op : OPÉRATEUR_DE_COMPARAIISON

Nature

Garde.

Utilisation pratique

Pour comparer deux nombres.

Évaluation

La garde est un SUCCES lorsque *m* et *n* sont deux NOMBRES reliés par la relation spécifiée. On rappelle que les OPÉRATEURS_DE_COMPARAIISON sont les suivants :

- Égal : =
- Différent : / =
- Inférieur : <
- Inférieur ou égal : <=
- Supérieur : >
- Supérieur ou égal : >=

Lorsque l'opérateur est celui d'égalité ou d'inégalité, la garde est aussi un SUCCES lorsque *m* et *n* sont deux IDENTIFICATEURS reliés par la relation spécifiée.

Exemple

```
THEORY ess IS
```

```
    btest(bb/=aa) &  
    bcall(WRITE: bwritef("test3: SUCCES\n"))  
=>  
    test3;
```



```
    bnot(btest(8=aa)) &
    bcall(WRITE: bwritef("test2: ECHEC\n")) &
    test3
=>
    test2;

    btest(8=8) &
    bcall(WRITE: bwritef("test1: SUCCES\n")) &
    test2
=>
    coco

END
```

Résultat

```
test1: SUCCES
test2: ECHEC
test3: SUCCES
```

Chapitre 74

bunify

$\text{bunify}(x, p, y, q, e, f)$

Paramètres

x : VARIABLE

p : FORMULE

y : VARIABLE

q : FORMULE

e : FORMULE

f : FORMULE

Nature

Garde.

Utilisation pratique

Pour vérifier que l'on peut unifier deux formules.

Évaluation

Pour que la garde soit un SUCCES, il est tout d'abord nécessaire que les formules p et q ne contiennent pas de quantificateurs. Il faut ensuite qu'il existe deux formules g et h telles que le remplacement de x par g dans p soit identique au remplacement de y par h dans q . Il faut, enfin, que ces deux formules g et h coïncident respectivement avec e et f (supposées non instanciées). La plupart du temps, e et f sont de simples jokers. Les jokers non instanciés de e et f sont alors instanciés.

Exemple

THEORY ess IS

```
bunify(X,P,Y,Q,E,F) &
bguard(WRITE: bwritef("\nOK\n%\n%\n%:=%\n%:=%\n\n",P,Q,X,E,Y,F)) &
bguard((SUB;RES): bresult([X:=E]P),A) &
bguard((SUB;RES): bresult([Y:=F]Q),B) &
bguard(WRITE: bwritef("A=%\nB=%\n",A,B))
```

```

=>
unify(X,P,Y,Q);

unify(x,    p(a,x    ,f(g(y))),
      (z,w),p(z,h(z,w),f(w    )))
=>
ccc;

bcall(ess: ccc)
=>
coco

```

END

Résultat

```

OK
p(a,x,f(g(y)))
p(z,h(z,w),f(w))
x:=h(a,g(y))
z,w:=a,g(y)

A=p(a,h(a,g(y)),f(g(y)))
B=p(a,h(a,g(y)),f(g(y)))

```

Chapitre 75

bunmask

bunmask

Paramètres

Aucun paramètre.

Nature

Garde.

Utilisation pratique

Annuler l'effet d'un **bmask** et exécuter l'éventuelle interruption mémorisée.

Évaluation

L'évaluation de la garde **bmask** est un succès si elle a lieu dans une portion non déjà masquée. De même, l'évaluation de la garde **bunmask** est un succès seulement si elle a lieu dans un portion masquée.

A partir de l'exécution de **bmask**, la réception des signaux SIGTERM est masquée jusqu'au prochain **bunmask**. On définit ainsi une portion masquée pendant laquelle la réception d'un signal SIGTERM est sans effet, elle est simplement mémorisée. Si plusieurs signaux SIGTERM sont reçus dans cette portion masquée, ils sont mémorisés comme une seule interruption. Lors de l'exécution de **bunmask**, le signal mémorisé a l'effet normal d'un SIGTERM : armement de la garde **binter** et échec de l'éventuelle sous preuve lancée par **bguardi**.

Exemple

```
THEORY start IS
```

```
    bcall((toto;ARI~): coco) => p
```

```
END
```

```
&
```

```
THEORY toto IS
```

```

bcall(WRITE: bwritef("bad interrupt memorisation\n"))
=>
coco2;

binter &
bcall(WRITE: bwritef("OK interrupt found\n"))
=>
coco2;

bunmask &
bcall(WRITE: bwritef("bad bunmask eval\n"))
=>
coco2;

bcall(WAIT: bwait(1)) &
coco(x+1)
=>
coco(x);

bmask &
bcall(WRITE: bwritef("bad bmask eval\n"))
=>
coco(x);

bunmask &
bcall(WRITE: bwritef("unmasking...\n")) &
coco2
=>
coco(10);

bmask &
bcall(WRITE: bwritef("masking, counting 10...\n")) &
coco(1)
=>
coco

```

END

Résultat

```

krt -c ess
krt -b ess.kin ess.ex &
[pid = 123]
masking, counting 10...
kill -15 123
(attente...)
unmasking...
OK interrupt found

```

Chapitre 76

bvrb

$\text{bvrb}(f)$

Paramètres

f : FORMULE.

Nature

Garde.

Utilisation pratique

Pour tester qu'une formule est une VARIABLE.

Évaluation

SUCCES si la formule f est une VARIABLE. On rappelle qu'une VARIABLE est soit une LETTRE, soit un IDENTIFICATEUR ne commençant pas par un UNDERSCORE, soit l'une des deux possibilités précédentes suivi d'un DOLLAR et d'un nombre inférieur à 10000, soit, enfin, une liste constitués d'éléments distincts et appartenant aux trois possibilités précédentes.

Exemple

THEORY ess IS

```
bcall(WRITE: bwritef("test5: ECHEC\n"))
=>
test5;

bvrb(a+b) &
bcall(WRITE: bwritef("test5: SUCCES\n"))
=>
test5;

bcall(WRITE: bwritef("test4: ECHEC\n")) &
test5
=>
```

```

    test4;

    bvrb(_a) &
    bcall(WRITE: bwritef("test4: SUCCES\n")) &
    test5
=>
    test4;

    bcall(WRITE: bwritef("test3: ECHEC\n")) &
    test4
=>
    test3;

    bvrb(a$10000) &
    bcall(WRITE: bwritef("test3: SUCCES\n")) &
    test4
=>
    test3;

    bcall(WRITE: bwritef("test2: ECHEC\n")) &
    test3
=>
    test2;

    bvrb(a,a,bbb) &
    bcall(WRITE: bwritef("test2: SUCCES\n")) &
    test3
=>
    test2;

    bcall(WRITE: bwritef("test1: ECHEC\n")) &
    test2
=>
    test1;

    bvrb(aaaa_3,xxx,xx$2,y) &
    bcall(WRITE: bwritef("test1: SUCCES\n")) &
    test2
=>
    test1;

    bcall(ess: test1)
=>
    coco

END

```

Résultat

test1: SUCCES
test2: ECHEC
test3: ECHEC
test4: ECHEC
test5: ECHEC

Chapitre 77

bwait

`bwait(n)`

Paramètres

n : NOMBRE

Nature

Opération sur un but

Utilisation pratique

Suspendre la preuve pendant *n* secondes, ou jusqu'à l'arrivée d'un signal.

Évaluation

La garde est un SUCCES lorsque *n* est un nombre. L'évaluation de la garde dure *n* secondes, sauf si un signal a été détecté, dans ce cas L'évaluation se termine à la réception. Si *n* est zéro, l'attente est infinie. Attention, le seul effet du signal reçu dans ce cas est de faire sortir de **bwait**. Par exemple, le signal reçu pendant un **bwait** dans une sous preuve lancée par **bguardi** n'arrête pas cette sous preuve.

Sous *UNIX*, tous les signaux interrompent **bwait**. Néanmoins, beaucoup de signaux provoqueront l'arrêt brutal de la tâche, c'est pourquoi il est conseillé d'utiliser **SIGHUP**, qui interrompt **bwait** sans modifier la preuve. **bwait** permet de libérer la machine pour l'exécution d'autres tâches simultanées.

Tactique

WAIT

Exemple

THEORY ess IS

```
bkid(n) &
bconnect("toto") &
bcall(WRITE: bprintf("%", n)) &
bclose &
bcall(WAIT: bwait(2)) &
bcall(WRITE: bwritef("fin 1er bwait\n")) &
```

```
bcall(WAIT: bwait(0)) &  
bcall(WRITE: bwritef("fin 2eme bwait\n"))  
=>  
P
```

END

Résultat

```
mach% krt -b ess.kin ess.ex &  
fin 1er bwait  
mach% kill -HUP 'cat toto'  
fin 2eme bwait  
done krt -b ess.kin ess.ex  
mach%
```

Chapitre 78

bwritef

`bwritef(f)` `bwritef(f, l)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_VIRGULES

Nature

Opération sur un but

Utilisation pratique

Écriture formatée sur le fichier standard de sortie

Évaluation

Chaque caractère de *f* (éventuellement ré-interprété) est écrit sur le fichier standard de sortie.

- **Interprétation des caractères** : Les caractères sont normalement interprétés littéralement. Seuls les caractères ou groupes de caractères qui suivent sont interprétés de façon particulière :
 - `\B` est la sonette
 - `\E` est l'échappement
 - `\t` est la tabulation
 - `\n` est le saut de ligne
 - `\"` est `"`
 - `\\` est `\`
 - `\%` est `%`
- `%`, non précédé de `\` et non suivi d'une chaîne numérique. L'interprétation est différente suivant la nature de l'opération :
 - `bwritef(f)` : aucune impression n'est effectuée
 - `bwritef(f, l)` : la $n^{\text{ème}}$ occurrence de `%` est remplacée par la formule de rang $((n - 1) \bmod t)$ de *l* (*t* est la longueur de la liste *l*).
- `%`, non précédé de `\` et suivi d'une chaîne numérique désignant un nombre *m*. L'interprétation est la même que dans le cas précédent sauf lorsque la formule à imprimer est un `NOMBRE` ou un `IDENTIFICATEUR`. Dans chacun de ces deux cas,

on imprime au moins m caractères ; pour cela, lorsque la formule à imprimer ne contient pas assez de caractères, on insère, en tête de la dite formule, autant de caractères blancs qu'il est nécessaire pour obtenir exactement le nombre de caractères requis.

Tactique

WRITE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
bcall(WRITE: bwritef("\BA%A%2A\t345\taaa\t\" \\ \% \tbbb\nBBB\n")) &
bcall(WRITE: bwritef("plus1: % +++ plus2: %\n",a+b)) &
bcall(WRITE: bwritef("res:%7 ^^^ %4 ^^^ %1 ^^^ %2 ^^^\n",250,aaa))
=>
coco
```

END

Résultat

```
AAA 345 aaa " \ % bbb
BBB
```

```
plus1: a+b +++ plus2: a+b
```

```
250 ^^^ aaa ^^^ 250 ^^^ aaa ^^^
```

Chapitre 79

bwritem

`bwritem(f)` `bwritem(f, l)`

Paramètres

f : CHAINE_ENTRE_GUILLEMETS

l : LISTE_DE_FORMULES_SÉPARÉES_PAR_DES_VIRGULES

Nature

Opération sur un but

Utilisation pratique

Écriture formatée sur la fenêtre de menu

Évaluation

Chaque caractère de *f* (éventuellement ré-interprété) est écrit sur la fenêtre de menu ou, à défaut, sur le fichier standard de sortie.

- **Interprétation des caractères** : Les caractères sont normalement interprétés littéralement. Seuls les caractères ou groupes de caractères qui suivent sont interprétés de façon particulière :
 - `\B` est la sonette
 - `\E` est l'échappement
 - `\t` est la tabulation
 - `\n` est le saut de ligne
 - `\"` est "
 - `\\` est \
 - `\%` est %
- `%`, non précédé de `\` et non suivi d'une chaîne numérique. L'interprétation est différente suivant la nature de l'opération :
 - `bwritem(f)` : aucune impression n'est effectuée
 - `bwritem(f, l)` : la $n^{\text{ème}}$ occurrence de `%` est remplacée par la formule de rang $((n - 1) \bmod t)$ de *l* (*t* est la longueur de la liste *l*).
- `%`, non précédé de `\` et suivi d'une chaîne numérique désignant un nombre *m*. L'interprétation est la même que dans le cas précédent sauf lorsque la formule à imprimer est un NOMBRE ou un IDENTIFICATEUR. Dans chacun de ces deux cas,

on imprime au moins m caractères ; pour cela, lorsque la formule à imprimer ne contient pas assez de caractères, on insère, en tête de la dite formule, autant de caractères blancs qu'il est nécessaire pour obtenir exactement le nombre de caractères requis.

Tactique

WRITE

Recul

L'effet de bord du à cette opération n'est pas détruit lors d'un recul

Exemple

THEORY ess IS

```
bcall(WRITE: bwrite("\BA%A%2A\t345\taaa\t\" \\ \%tbbb\nBBB\n\n")) &
bcall(WRITE: bwrite("plus1: % +++ plus2: %\n\n",a+b)) &
bcall(WRITE: bwrite("res:%7 ^^^ %4 ^^^ %1 ^^^ %2 ^^^\n",250,aaa))
=>
coco
```

END

Résultat

```
AAA 345 aaa " \ % bbb
BBB
```

```
plus1: a+b +++ plus2: a+b
```

```
250 ^^^ aaa ^^^ 250 ^^^ aaa ^^^
```