Référence

Quadruplets

ADD (*d*, *a*, *b*) - *d* ← *a* + *b*
affecte à la variable *d* la somme des variables *a* et *b*.

CALL (*n*) - S ←empiler (S, PC + 1); PC ← *n*
réalise un appel au sous-programme dont le premier quadruplet a pour numéro n.

DIV (*d*, *a*, *b*) - *d* ← *a* / *b*
affecte à la variable *d* le quotient entier de la variable *a* par la variable *b*.

GOTO (*n*) - PC ← *n*
réalise un branchement, l'exécution continue au quadruplet *n*.

GOTO_EQ (*n*, *a*, *b*) - si *a* = *b* alors PC ← *n*
réalise le branchement sur le quadruplet *n* si *a* = *b*.

GOTO_GE (*n*, *a*, *b*) - si *a* ≥ *b* alors PC ← *n*
réalise le branchement sur le quadruplet *n* si *a* ≥ *b*.

GOTO_GT (*n*, *a*, *b*) - si *a* > *b* alors PC ← *n*
réalise le branchement sur le quadruplet *n* si *a* > *b*.

GOTO_LE (*n*, *a*, *b*) - si *a* ≤ *b* alors PC ← *n*
réalise le branchement sur le quadruplet *n* si *a* ≤ *b*.

GOTO_LT (*n*, *a*, *b*) - si *a* < *b* alors PC ← *n*
réalise le branchement sur le quadruplet *n* si *a* < *b*.

GOTO_NE (*n*, *a*, *b*) - si *a* ≠ *b* alors PC ← *n*
réalise le branchement sur le quadruplet *n* si *a* ≠ *b*.

INVOKE (*d*, *a*, *b*) - spécial
réalise une sorte d'*appel système* auprès de la machine virtuelle; celui-ci est passé au module réalisant l'application Karel. La signification des paramètre *d*, *a* et *b* est dépendante de l'application sous-jacente.

MUL (*d*, *a*, *b*) - *d* ← *a* × *b*
affecte à la variable *d* le produit des variables *a* et *b*.

NAND (*d*, *a*, *b*) - *d* ← ¬(*a* ∧ *b)*
affecte à la variable *d* le NON-ET logique de la variable *a* et de la variable *b*.

RETURN - PC ← dépiler(S)
réalise un retour de sous-programme en dépilant de S le numéro du quadruplet à exécuter.

SET (*d*, *a*) - *d* ← *a*
affecte à la variable *d* la valeur contenue dans la variable *a*.

SETI (*d*, *a*, *b*) - *d* ← valeur(*a*)
affecte à la variable *d* la valeur *a*.

STOP - arrêt
provoque l'arrêt de la machine virtuelle.

SUB (*d*, *a*, *b*) - *d* ← *a* - *b*
affecte à la variable *d* la différence des variables *a* et *b*.

Commandes de Karel

`Karel.any_beeper` (*a* = numéro de variable, *b* non utilisé)
Renvoie 1 dans la variable de numéro *b* s'il y a encore des beepers dans le panier de Karel, 0 sinon.

`Karel.facing` (*a* = orientation, *b* = numéro de variable)
Renvoie 1 dans la variable de numéro *b* si le robot est orienté selon *a*, 0 sinon. *a* peut prendre les valeurs `Karel.north`, `Karel.east`, `Karel.south` ou `Karel.west`.

`Karel.is_blocked` (*a* = direction, *b* = numéro de variable)
Renvoie 1 dans la variable de numéro *b* s'il n'y a un mur dans la direction *a*, 0 sinon. *a* peut prendre les valeurs `Karel.front`, `Karel.left`, `Karel.right`.

`Karel.is_clear` (*a* = direction, *b* = numéro de variable)
Renvoie 1 dans la variable de numéro *b* s'il n'y a aucun mur dans la direction *a*, 0 sinon. *a* peut prendre les valeurs `Karel.front`, `Karel.left`, `Karel.right`.

`Karel.move` (*a*, *b* non utilisés)
Avancer en avant.

`Karel.next_beeper` (*a* = numéro de variable, *b* non utilisé)
S'il y a un beeper à la position courante, renvoie 1 dans la variable de numéro *a*, renvoie 0 sinon.

`Karel.no_beeper` (*a* = numéro de variable, *b* non utilisé)
Renvoie 1 dans la variable de numéro *b* s'il n'y a plus de beeper dans le panier de Karel, 0 sinon.

`Karel.no_next_beeper` (*a* = numéro de variable, *b* non utilisé)
S'il n'y a pas de beeper à la position courante, renvoie 1 dans la variable de numéro *a*, renvoie 0 sinon.

`Karel.not_facing` (*a* = orientation, *b* = numéro de variable)
Renvoie 1 dans la variable de numéro *b* si le robot est n'est pas orienté selon *a*, 0 sinon. *a* peut prendre les valeurs `Karel.north`, `Karel.east`, `Karel.south` ou `Karel.west`.

`Karel.pick_beeper` (*a*, *b* non utilisés)
Récupère un beeper de la position courante.

`Karel.put_beepr` (*a*, *b* non utilisés)
Dépose un beeper à la position courante.

`Karel.turn_left` (*a*, *b* non utilisés)
Tourner à gauche.

Primitives de compilation

```
(** Ajoute un nouveau quad au programme.
   @param quad         Quad ajouté. *)
let gen quad = ...

(** Obtient le numéro de quad suivant.
   @return    Numéro de quad suivant. *)
let nextquad _ = ...

(** Obtiens un nouveau numéro de variable temporaire.
   @return       Numéro de variable. *)
let new_temp _ = ...

(** Applique un backpatch sur le goto à l'adresse g pour qu'il branche
   à l'adresse a.
   @param g    Adresse du goto.
   @param a    Adresse de branchement. *)
let backpatch g a = ...

(** Test si le paramètre id est défini ou non.
   @parm id    Identificateur à tester.
   @return       Vrai si le symbole existe, faux sinon. *)
let is_defined id = ...

(** Permet de définir une nouvelle définition.
   @param id    Identificateur de la définition.
   @param addr    Adresse du sous-programme. *)
let define id addr = ...

(** Renvoie l'adresse du sous-programme associé avec l'identificateur.
   @param id    Identifiant du sous-programme.
   @return       Adresse du sous-programme. *)
let get_define id = ...
```