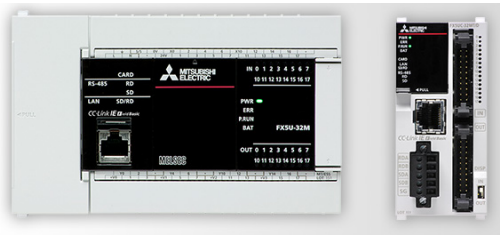
Des automates programmables industriels aux BDD paramétriques: de l'avantage de nuancer les décisions binaires

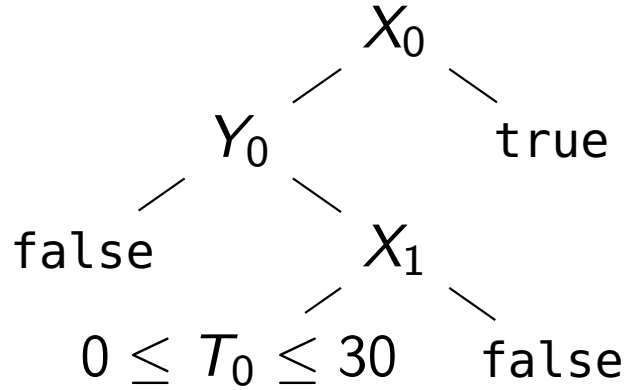
Claude Marché[1] Denis Cousineau[2]

[1]Université Paris-Saclay, Inria & LMF

[2]Mitsubishi Electric R&D Centre Europe, Rennes

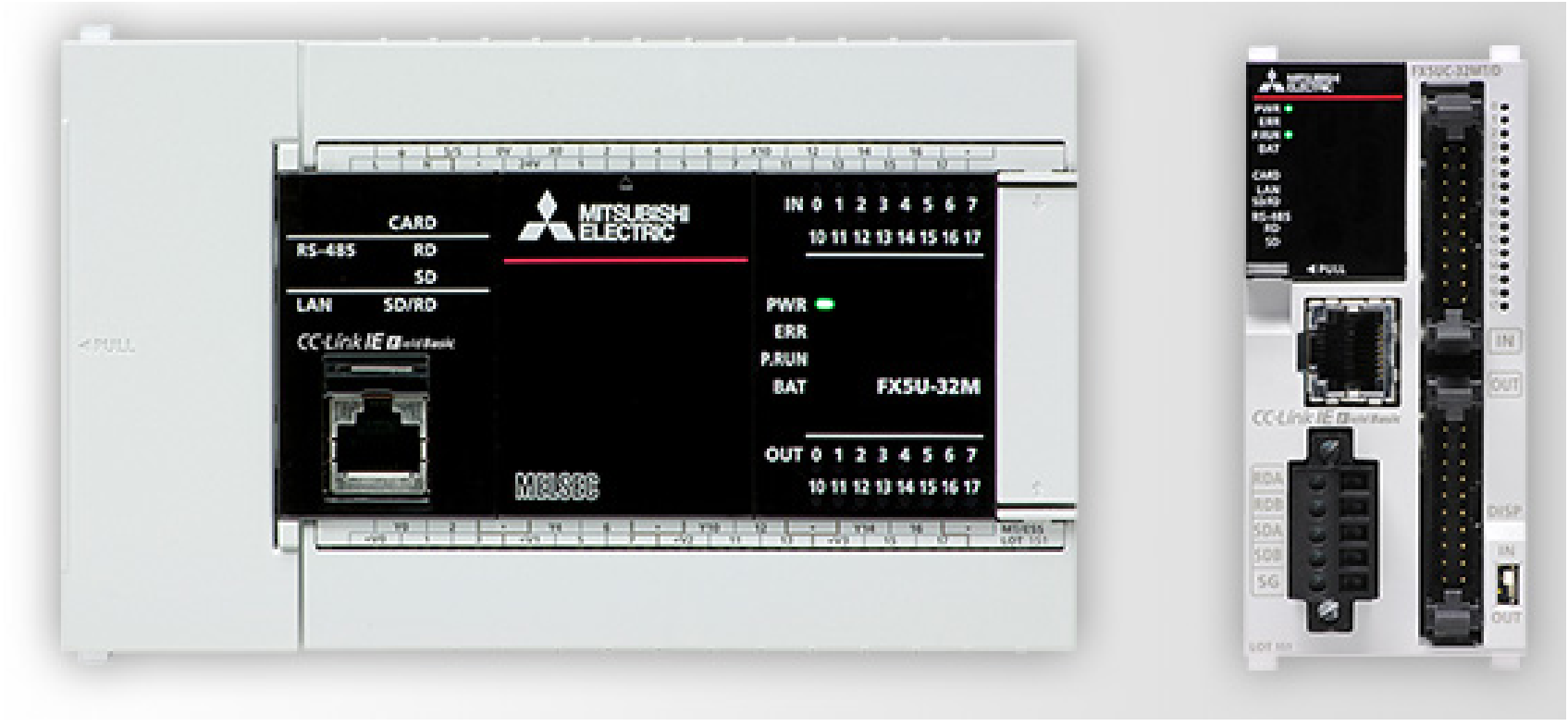
JFLA, 30 janvier 2024





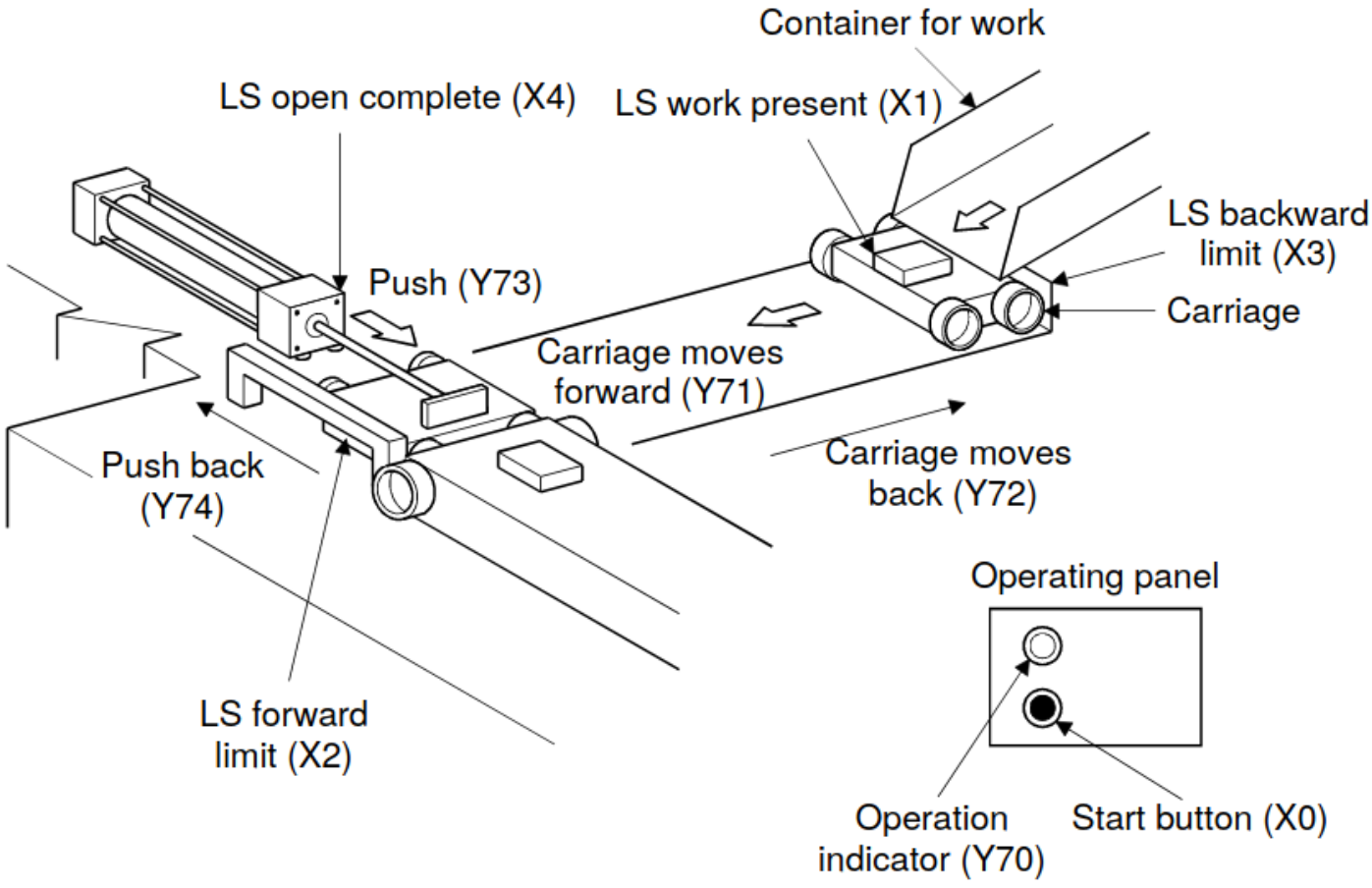
Contexte: Les Automates Programmables Industriels

(en anglais: *Programmable Logic Controller*, PLC)



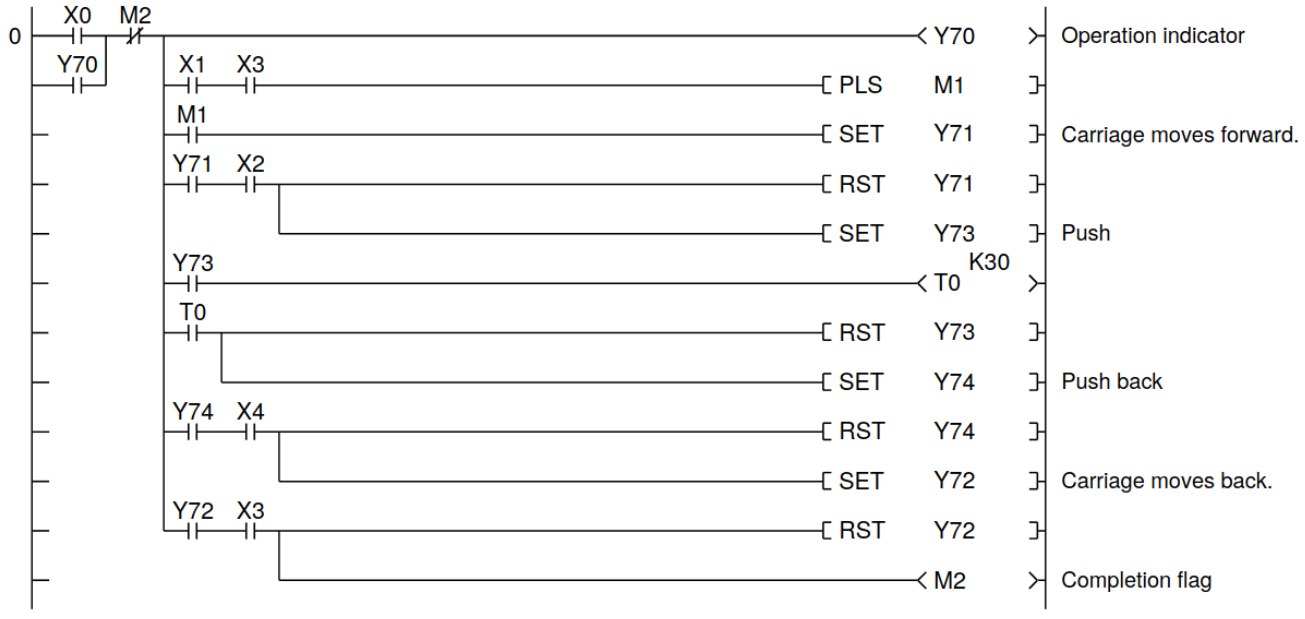
MELSEC IQ-F series

Exemple: *carriage line control*



$X_0,\ X_1,\ \ldots$: entrées $\qquad$ $Y_0,\ Y_1,\ \ldots$: sorties

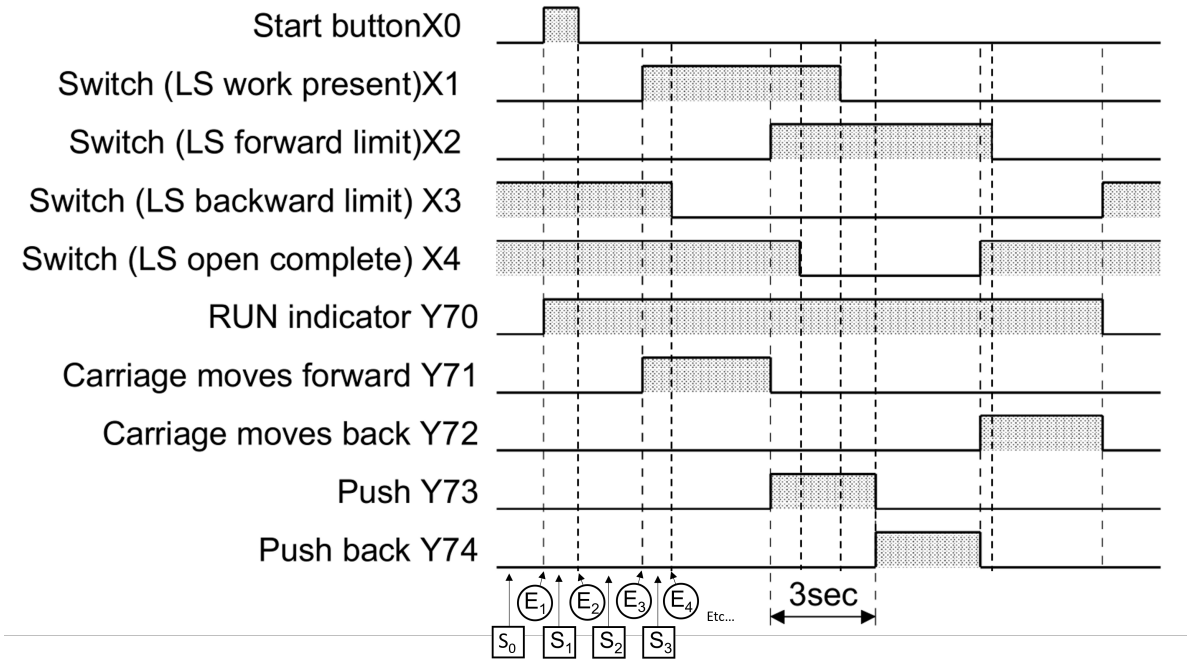
Programme Ladder



X_0, X_1, ..., Y_0, Y_1, ..., M_0, M_1, ... : valeurs booléennes

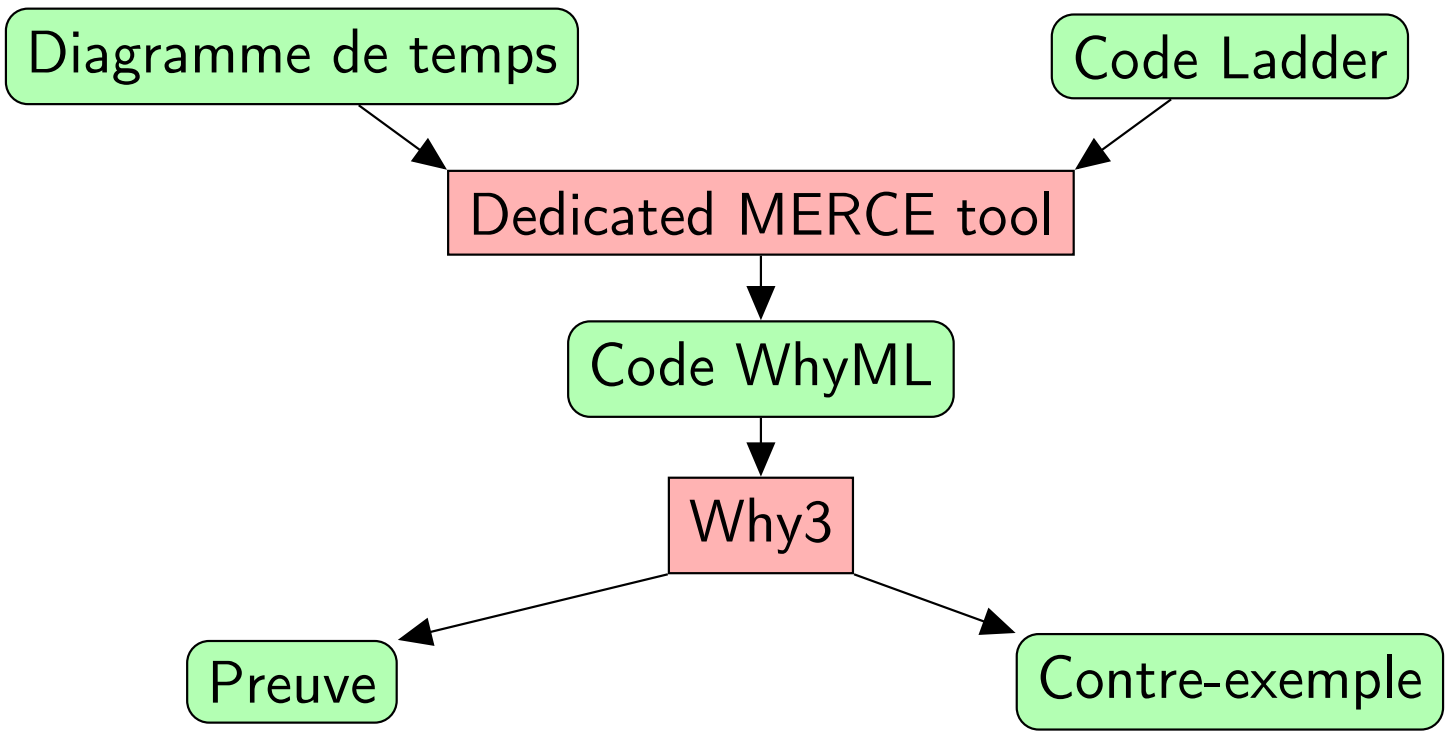
T_0 : valeur entière

Une spécification par *Diagramme de temps*



Analyse de conformité

Belo Lourenço et al., STTT 2022



Encodage vers WhyML: un scan

```
let scan () =
  let f1 = (!x0 || !y70)
        && (not !m2) in
  y70 := f1;
  let f2 = !x1 && !x3 in
  pls (f1 && f2) m1 cc0;
  let f3 = !m1 in
  set (f1 && f3) y71;
  let f4 = !y71 && !x2 in
  rst (f1 && f4) y71;
  set (f1 && f4) y73;
  let f5 = !y73 in
  timer_coil (f1 && f5) t0 30;
  let f6 = timer_contact t0 in
  rst (f1 && f6) y73;
  set (f1 && f6) y74;
  let f7 = !y74 && !x4 in
  rst (f1 && f7) y74;
  set (f1 && f7) y72;
  let f8 = !y72 && !x3 in
  rst (f1 && f8) y72;
  m2 := f1 && f8
```

- Généré automatiquement depuis le code Ladder
- Beaucoup de variables *booléennes*
- Quelques variables *entières*
- Appel de *fonctions auxiliaires* avec effets de bord

Encodage vers WhyML: itérations

```
let carriage ()
  requires { ... }
= scan ();
  assert { A0 };
  while C1 do invariant { I1 }
    scan ()
  done;
  scan();
  assert { A1 };
  while C2 do invariant { I2 }
    scan ()
  done;
  ⋮
  while C10 do invariant { I10 }
    scan ()
  done;
  scan()
  assert { A10 };
```

- Généré automatiquement depuis le diagramme de temps
- 10 boucles en séquence
- Besoin *d'invariants de boucle supplémentaires* pour prouver un tel programme

Analyse de conformité: résumé

Belo Lourenço et al., STTT 2022

- Besoin de *générer des invariants de boucle*
- Fait par un interpréteur abstrait prototype de Why3 (InferLoop) utilise la libraire Apron et des domaines ad-hoc [Baudin 2016]...
- ... étendu pour le support des booléens [Belo Lourenço 2022]
- Résultats: analyse automatique fonctionne mais *temps de calcul décevant*

Objectif

Améliorer notre interpréteur abstrait

Overview

Cas des programmes purement booléens

Programmes mélangeant entiers et booléens

Évaluation sur codes issus de Ladder

Conclusions

Overview

Cas des programmes purement booléens

Programmes mélangeant entiers et booléens

Évaluation sur codes issus de Ladder

Conclusions

Programmes booléens

Interprétation abstraite dédiée aux programmes avec

- uniquement des variables booléennes
- des appels de fonctions

Domaine abstrait candidat

les *diagrammes de décision binaire* (BDD)

- un BDD représente naturellement un ensemble d'états pour les variables du programme

Bibliothèque OCaml candidate

OCaml-BDD [Filliâtre]

`https://github.com/backtracking/ocaml-bdd`

OCaml-BDD en bref

```
type variable = int
type t
val zero : t (* i.e. false *)
val one : t (* i.e. true *)
val mk_var : variable -> t
val mk_not : t -> t
val mk_and : t -> t -> t
val mk_or : t -> t -> t
val is_sat : t -> bool
```

- Interface purement fonctionnelle
- Implémentation en très bref:

  ```
  type t = Zero | One | Node of variable * t * t
  ```

  avec *partage maximal* (hash-consing)

  ```
  let is_sat x = x <> Zero
  ```

Interprétation abstraite avec BDD

Conditionnelles, boucles

gérées grâce aux conjonctions et disjonctions

Affectations, appels de fonctions

besoin de

- renommage
- élimination existentielle

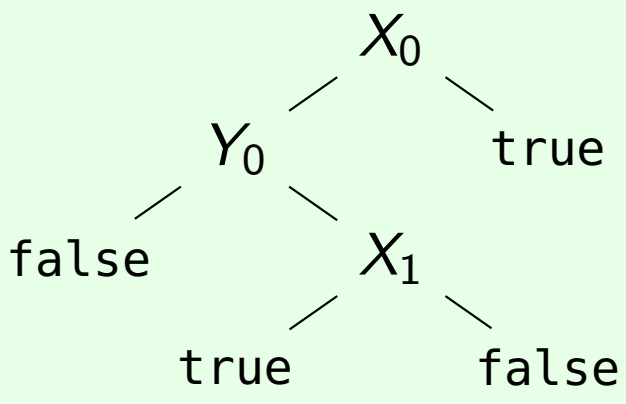
Restitution des invariants

besoin de traduction d'un BDD vers une formule « lisible »

Nos contributions à OCaml-BDD

- Fonctions d'élimination de quantificateurs
- Fonctions de renommage
- Reconstruction d'une formule logique *compacte* à partir d'un BDD

Exemple de reconstruction



$\rightarrow \quad X_0 \vee (Y_0 \wedge \neg X_1)$

Exemple

```
val random_bool () : bool
val a : ref bool
val b : ref bool
val c : ref bool

let f () =
  a := False; b := False; c := False;
  while not !c do
    if !b then c := True;
    if !a then b := True;
    let x = random_bool () in if x then a := True;
  done;
  assert { !a }
```

Invariant de boucle généré

$$\texttt{if } a \texttt{ then } (b \vee \neg c) \texttt{ else } (\neg b \wedge \neg c)$$

Overview

Cas des programmes purement booléens

Programmes mélangeant entiers et booléens

Évaluation sur codes issus de Ladder

Conclusions

Nuancer les décisions binaires

Idée centrale

Aux feuilles des BDD, au lieu de mettre seulement `true` ou `false`, on met une contrainte arbitraire

$\mapsto$ Besoin d'une nouvelle structure de données: *BDD paramétriques*

Difficultés

- Conserver le partage maximal
- Le domaine paramètre peut être « contextuel »

BDD paramétriques

```
type t = Zero | One | Node of variable * t * t | Leaf of int
```

- L'implémentation maintient un partage maximal des arbres de type `t`.
- L'interprétation d'une feuille `Leaf k` dépend d'un *contexte* fournit par le domaine paramètre

Exemple

Opération meet

(conjonction)

```
let meet ctx b1 b2 =
  let cache = H2.create cache_default_size in
  let rec app ((u1,u2) as u12) = if u1 == u2 then u1 else
   try H2.find cache u12 with Not_found ->
     let res = match u1.node,u2.node with
     | Leaf i, Leaf j ->
       begin match D.meet ctx i j with
         | Some k -> mk_leaf k (* [Leaf k] with hash-consing *)
         | None -> zero
       end
     | ...
     in
     H2.add cache u12 res; res
  in
  app (b1, b2)
```

Le foncteur `BDDparam`

(très court extrait)

```
module type ParamDomain = sig
  type p_index = int
  type p_context
  val meet : p_context -> p_index -> p_index -> p_index option
  (** when result is [None], it means [bottom] *)
  ⋮
end

module Make (D: ParamDomain) : sig
  type t
  val meet : D.p_context -> t -> t -> t
  ⋮
end
```

Une instance: contraintes linéaires sur les entiers

Utilisation de la bibliothèque Apron

```
type p_state = Polka.strict Polka.t Abstract1.t
type p_context = {
  apron_env : Environment.t;
  mutable counter : int;
  state_to_int : int H.t;
  int_to_state : p_state Hint.t;
}

let meet ctx i j =
  let si = get ctx i in let sj = get ctx j in
  let s = Abstract1.meet man si sj in
  if Abstract1.is_bottom man s then None else Some (record ctx s)
```

Remarque

C'est l'instance qui gère la numérotation/mémoïsation des états abstraits numériques. Cette numérotation dépend du contexte.

Exemple (représentatif des codes issus de Ladder)

```
let f ()
  requires { s0 ∧ not s1 ∧ c = 0 }
= while True do
    if !s1 then c := !c +1;
    let tmp = random_bool () in
    if !s0 && tmp then (s0 := False; s1 := True; c := 0)
    else if (!s1 && (!c > 42)) then (s1 := False; break)
  done;
  assert { !c ≥ 43 }
```

Invariant généré

$$\texttt{if } s_1 \texttt{ then } \neg s_0 \wedge 0 \leq c \leq 42 \texttt{ else } s_0 \wedge c = 0$$

Gain de précision

Notre ancienne implémentation InferLoop ne génère pas un invariant assez précis pour prouver l'assertion

Overview

Résultats expérimentaux

Exemple: *Carriage line control*

- Partie supérieure: chaque boucle dans un code WhyML distinct
- Partie inférieure: exemple complet
- InferLoop: ancienne implémentation, BDDinfer: nouvelle implémentation

év./état	1	2	3	4	5	6	7	8	9	10
InferLoop	1.02	0.95	2.05	0.97	5.45	1.49	1.21	1.44	1.29	0.86
BDDinfer	0.12	0.10	0.12	0.10	0.33	0.10	0.09	0.13	0.12	0.08

év./états	1	1→2	1→3	1→4	1→5	1→6	1→7	1→8	1→9	1→10
InferLoop	1.02	33.5	292	1089	-	-	-	-	-	-
BDDinfer	0.12	0.19	0.30	0.42	0.53	0.91	1.00	1.10	1.22	1.35
# vars	47	95	143	191	239	287	335	379	427	475

(temps d'exécution en secondes)

Overview

Conclusions

Contributions

- Extensions de la bibliothèque OCaml-BDD
- Version fonctorisée `BDDparam`, instanciée avec Apron
- Interpréteur abstrait utilisant `BDDparam`
- Disponible dans Why3
- Gain très significatif en terme de temps de calcul et de précision

Travaux connexes

- Insuccès de l'utilisation de BDD-Apron, de Mopsa
- Articles discutés dans les actes

Perspectives

Au-delà des entiers et des booléens

- domaine de contraintes sur des symboles non-interprétés
- structure de données candidate: E-graph

Application à Ladder

- formes de spécifications temporelles plus complexes

Interprétation abstraite

- BDD paramétrés alternative intéressante pour gérer les disjonctions ?