Septième partie

Vérification de propriétés

Objectif

Vérifier si un système possède une propriété donnée : $\mathcal{M} \models P$.
Si le système et la propriété sont dans le même formalisme (cas de TLA$^+$), cela revient à vérifier si $\mathcal{M} \Rightarrow P$.

Plan

Principe

Le principe de la vérification est le parcours systématique et éventuellement exhaustif de l'espace d'états du système, afin de vérifier la compatibilité avec les propriétés attendues.
$\rightarrow$ système de transition fini.

- Génération en avant : depuis les états initiaux, pour construire le graphe d'états et vérifier tout type de propriété temporelle (cas de TLA$^+$)
- Génération en arrière :
  - Depuis un (ensemble d')état(s) dont on veut vérifier l'accessibilité depuis un état initial
  - depuis les états interdits (illégaux), pour vérifier qu'ils sont inaccessibles depuis un état initial (propriété de sûreté)

Analyse du problème

Pour construire le graphe d'états, on a besoin de :

- mémoriser les états déjà visités ($\mathcal{A}nciens$) ;
- mémoriser les nouveaux états à explorer ($\mathcal{N}ouveaux$) : états successeurs ou prédécesseurs des états visités (selon que l'on construit le graphe en avant depuis les états initiaux, ou en arrière depuis des états dit terminaux, par exemple les états interdits).

L'ensemble des états visités finit par devenir beaucoup plus gros que l'ensemble des états à explorer, par simple accumulation : les ensembles $\mathcal{A}$ et $\mathcal{N}$ n'auront pas nécessairement la même représentation mémoire pour des raisons d'efficacité.

Génération de modèle – TLA$^+$

Système $=$ *Init* $\wedge \Box[$*Next*$]$

```
procedure MC(Init, Next)
begin
  A := {};   -- états explorés
  N := Init; -- états nouveaux, à explorer
  while (N ≠ {})
  begin
   state  := pop(N);      -- choix d'un état à explorer
   succ := state ∧ Next; -- états successeurs
   A := A ∪ {state};      -- state a été traité
   N := N ∪ (succ \ A);  -- images nouvelles à explorer
  end;
  return A;               -- tout a été exploré
end
```

Algorithme générique

```
procedure exploration(Init, Image, Pop, Push, Minus, Union)
begin
  A := {}; -- états explorés
  N := Init; -- états non explorés
  while (N ≠ {})
  begin
   choix  := Pop(N);                    -- choix d'état(s) à explorer
   images := Image(choix) Minus A;      -- Images non déjà explorées
   A := A Union choix;                  -- choix a été traité
   N := Push(N, images);                -- ses images sont à explorer
  end;
  return A;                             -- tout a été exploré
end
```

Prérequis pour l'efficacité

États déjà visités

- Représentation concise en mémoire (très gros ensemble).
- Opérations pour l'ajout d'états nouveaux (`Union`).

États nouveaux

- Opération pour la différence ensembliste avec les états déjà visités (`Minus`) ;
- Opération pour le calcul d'image directe ou inverse (`Image`) ;
- Représentation mémoire permettant de construire un ordre global (`Pop` et `Push`).

Quelques solutions possibles

Représentation des états déjà visités

- explicite : bit-state encoding, table de hachage (risque de collision).
- implicite ou symbolique : Binary Decision Diagrams, formules SAT, polyhèdres...
- mixte : explicite + fingerprint (TLA$^+$)

Ordre global sur les états nouveaux

- parcours en profondeur (occupation mémoire limitée)
- parcours en largeur (petits contre-exemples)

Plan

Syntaxe alternative (rappel)

Syntaxe alternative

A (all)	$\leftrightarrow \forall$
E (exists)	$\leftrightarrow \exists$
G (globally)	$\leftrightarrow \Box$
F (finally)	$\leftrightarrow \Diamond$
X (next)	$\leftrightarrow \bigcirc$
A(f U g) ou f AU g	$\leftrightarrow f \forall \mathcal{U} g$
E(f U g) ou f EU g	$\leftrightarrow f \exists \mathcal{U} g$

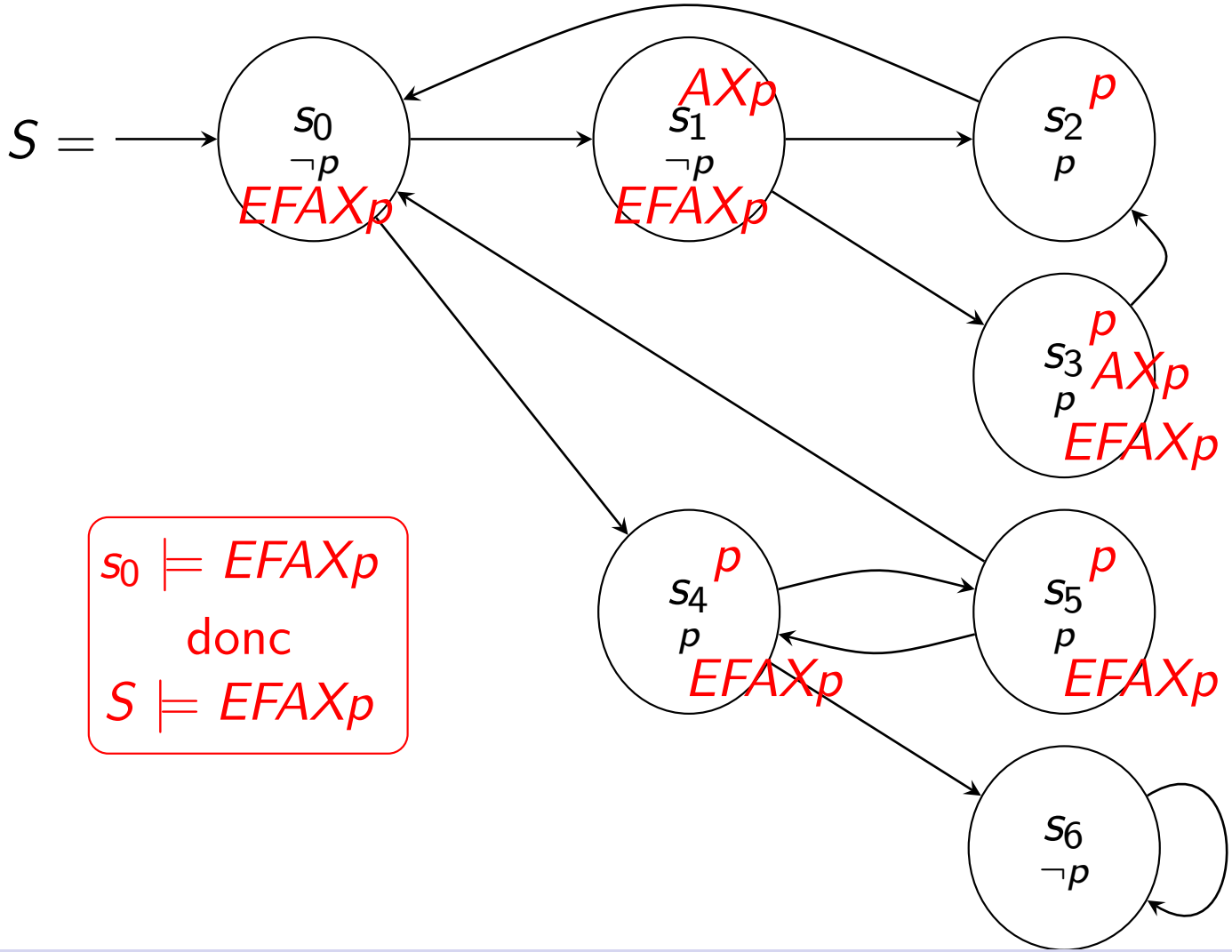
Par exemple : AG EF f $\leftrightarrow \forall \Box \exists \Diamond f$

Principe

- Les modèles en CTL sont les états (et non les exécutions).
- Calcul de l'ensemble des états qui satisfont une formule donnée, de façon inductive sur la structure de la formule.
- On procède par marquage des états :
  $s \mapsto \{F \mid s \models F\}$ (plutôt que $F \mapsto \{s \mid s \models F\}$).
- Chaque opérateur d'une formule F peut amener un parcours complet de S.

Marquage : un exemple

$EF\ AX\ p =$ existence d'un état dont tous les successeurs vérifient p.



Marquage CTL : $mark(F)$

- Si $F = p$ prédicat d'état, marquer les états qui vérifient le prédicat :

  `pourtout s ∈ S : s.F := eval(s,p)`

  (où `eval(s,p)` permet d'évaluer un prédicat dans un état)
- Si $F = \neg F'$, marquer les états non marqués par F' :

  ```
  mark(F');
  pourtout s ∈ S : s.F := not s.F';
  ```
- Si $F = F_1 \wedge F_2$, marquer les états marqués par les deux formules :

  ```
  mark(F1);
  mark(F2);
  pourtout s ∈ S : s.F := s.F1 and s.F2;
  ```

Marquage CTL : $mark(F)$

- Si $F = EX\ H$, marquer les états qui peuvent atteindre H en une transition :

```
mark(H);
pourtout s ∈ S : s.F := false;
pourtout s ∈ S, pourtout t ∈ succ(s) :
    si t.H alors s.F := true;
```

- Si $F = AX\ H$, marquer les états qui atteignent H par toutes les transitions :

```
mark(H);
pourtout s ∈ S : s.F := false;
pourtout s ∈ S :
    si pourtout t ∈ succ(s), t.H alors
        s.F := true;
```

Marquage CTL : $mark(F)$

- Si $F = H\ EU\ K$, idée : $H\ EU\ K \equiv K \vee (H \wedge EX(H\ EU\ K))$

```
mark(H);
mark(K);
L := ∅;
pourtout s ∈ S :
    s.F := s.K;
    si s.F alors L := L ∪ {s};
tantque L≠ ∅ faire:
    s := choose any in L;  // ordre sans importance
    L := L \ {s};
    pourtout t ∈ pred(s) :
        si t.H et ¬t.F alors   // pas déjà marqué
            t.F = true;
            L := L ∪ {t};
        finsi
    finpour
fintq
```

Marquage CTL : $mark(F)$

- Si $F = H\ AU\ K$, idée : $H\ AU\ K \equiv K \vee (H \wedge AX(H\ AU\ K))$

```
mark(H); mark(K);
L := ∅;
pourtout s ∈ S :
    s.F := s.K; s.nb := card(succ(s));
    si s.F alors L := L ∪ {s};
tantque L≠ ∅ faire:
    s := choose any in L;  // ordre sans importance
    L := L \ {s};
    pourtout t ∈ pred(s) :
        t.nb := t.nb - 1;
        si t.nb = 0 et t.H et ¬t.F alors
            t.F := true;
            L := L ∪ {t};
        finsi
    finpour
fintq
```

CTL équitable

Fair CTL = CTL + des contraintes d'équité multiple sur les états.

- Contraintes d'équité : un ensemble d'ensemble d'états $F_1, \ldots, F_n$
- Trace équitable : trace qui visite infiniment souvent tous les F_i
- État équitable : état à partir duquel il existe au moins une trace équitable
- Avoir une propriété en CTL équitable :

$$M \models_f \phi \triangleq M \models \mathit{fair} \Rightarrow \phi \text{ avec } \mathit{fair} = \bigwedge_{1 \leq i \leq n} \Box\Diamond F_i$$

Ce n'est pas une formule de CTL (ni de LTL) !

CTL équitable – définition

Pour un état s, $s \models_f \phi$ signifie que s vérifie ϕ avec les contraintes d'équité :

- $s \models_f EX\phi$ s'il existe une trace équitable $s_0 \cdot s_1 \cdots$ avec $s_0 = s$ et tel que $s_1 \models_f \phi$.
- $s \models_f AX\phi$ si pour toute trace équitable $s_0 \cdot s_1 \cdots$ avec $s_0 = s$, $s_1 \models_f \phi$.
- $s \models_f EG\phi$ s'il existe une trace équitable $s_0 \cdot s_1 \cdots$ avec $s_0 = s$ et tel que $\forall i, s_i \models_f \phi$.
- $s \models_f AG\phi$ si pour toute trace équitable $s_0 \cdot s_1 \cdots$ avec $s_0 = s$, $\forall i, s_i \models_f \phi$.
- $s \models_f \phi EU\psi$ s'il existe une trace équitable $s_0 \cdots s_k \cdots$ avec $s_0 = s$ et tel que $s_k \models_f \psi$ et $\forall 0 \leq i < k, s_i \models_f \phi$
- $s \models_f \phi AU\psi$ si pour toute trace équitable $\exists k, s_0 \cdots s_k \cdots$ avec $s_0 = s$, et $s_k \models_f \psi$ et $\forall 0 \leq i < k, s_i \models_f \phi$

Réduction à CTL classique

On suppose qu'on a marqué les états avec un prédicat *fair* qui dit qu'il existe une trace équitable issue de l'état.

- $s \models_f EX\phi$ ssi $s \models EX(\phi \wedge fair)$
- $s \models_f AX\phi$ ssi $s \models AX(fair \Rightarrow \phi)$
- $s \models_f EG\phi$ est compliqué
- $s \models_f AG\phi$ ssi $s \models AG(fair \Rightarrow \phi)$
- $s \models_f \phi EU\psi$ ssi $s \models \phi EU(fair \wedge \psi)$
- $s \models_f \phi AU\psi$ ssi $s \models_f \neg EG\neg\psi$ et $s \models_f \neg((\neg\psi)EU(\neg\phi \wedge \neg\psi))$

Calcul des états équitables (*fair*)

Soit un graphe d'états S et un ensemble de contraintes d'équité F_i.

1. Calculer $SCC(S)$, l'ensemble des *strongly connected components* (composantes fortement connexes) du graphe S. (= les sous-graphes où l'on peut rester boucler à l'infini)
2. Calculer l'union des SCC qui intersectent tous les F_i :
   $S' = \bigcup_{C \in SCC(S)} (C \text{ si } \forall i : C \cap F_i \neq \emptyset)$
   (= le sous-graphe où l'on peut boucler en visitant infiniment tous les F_i)
3. États équitables : $S'' = \{u \in S : u \rightarrow^* v \text{ où } v \in S'\}$
   (fermeture réflexive transitive des prédécesseurs des états de $S' = S'$ et les états qui peuvent y conduire)
4. Chaque état de S'' est alors marqué *fair*.

Vérification de $S \models_f EG\phi$

On vérifie l'existence d'une trace vérifiant ϕ et conduisant à une (ou plusieurs) composante fortement connexe où ϕ est toujours vrai :

- Marquer les états avec ϕ
- Soit $S(\phi)$ la restriction du système aux états vérifiant ϕ
- Calculer $SCC(S(\phi))$
  (les sous-graphes où l'on boucle infiniment en conservant ϕ)
- Calculer S_f l'union des SCC qui intersectent tous les F_i
  (= le sous-graphe vérifiant toujours ϕ et où l'on peut visiter infiniment souvent tous les F_i)
- $s \models_f EG\phi$ ssi $s \models \phi \; EU \; S_f$
  (un préfixe vérifiant ϕ, suivi d'un suffixe infini vérifiant ϕ et visitant infiniment les F_i)

Complexité

- CTL classique : complexité linéaire en le nombre d'opérateurs $= O(|S| \times |\phi|)$.
- CTL équitable :
  - Calcul des SCC linéaire (algorithme de Tarjan)
  - Calcul d'accessibilité linéaire
  - Transformation linéaire en CTL classique
  - Complexité $= O(|S| \times |\phi|)$.

(rappel : on peut ramener l'équité sur les transitions WF(A) et SF(A) à l'équité multiple sur les états)

Plan

Réduction du problème

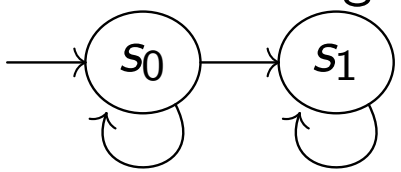
Dans un système ayant un nombre fini d'états, la seule solution pour obtenir une exécution infinie est de finir par boucler sur un cycle.
$\rightarrow$ On se ramène à un problème de traces ω-régulières : un préfixe puis un cycle (un lasso).

Traces ω-régulières

Soit $\mathcal{S} = \langle S, I, R \rangle$ un système de transition fini, L une formule LTL

$$\begin{aligned} & \exists \sigma \in S^{\omega} : \sigma \in \mathit{Exec}(\mathcal{S}) \wedge \sigma \models L \\ \Leftrightarrow \quad & \exists \sigma_p, \sigma_c \in S^{\star} : \langle \sigma_p \rightarrow {\sigma_c}^{\omega} \rangle \in \mathit{Exec}(\mathcal{S}) \wedge \langle \sigma_p \rightarrow {\sigma_c}^{\omega} \rangle \models L \end{aligned}$$

Même ainsi, il peut exister une infinité de traces ω-régulières : une infinité de préfixes distincts. Considérer



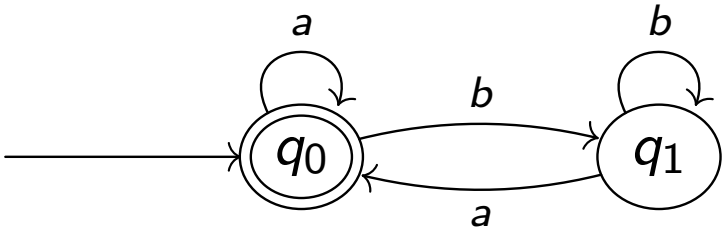
Automate de Büchi

Automate de Büchi

$A = (Q, \Sigma, \delta, q_0, F)$, où

- Q : ensemble fini d'états
- Σ : alphabet fini
- $q_0 \in Q$: l'état initial de l'automate
- $F \subseteq Q$: les états acceptants
- $\delta \in Q \times \Sigma \mapsto Q$: fonction de transition de l'automate.

Un mot infini est accepté si sa reconnaissance visite infiniment F.



reconnaît les mots infinis sur $\{a, b\}$ ayant une infinité de a.

Intérêt des automates de Büchi

Ensemble infini de traces

- Permet de représenter un ensemble infini de traces ω-régulières
- Suffisant pour représenter tout système de transition fini
- Suffisant pour représenter toute formule de LTL

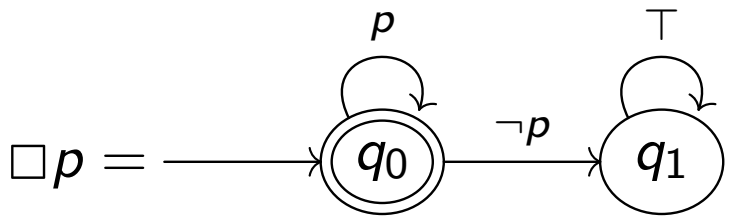
Manipulation d'ensembles infinis de traces

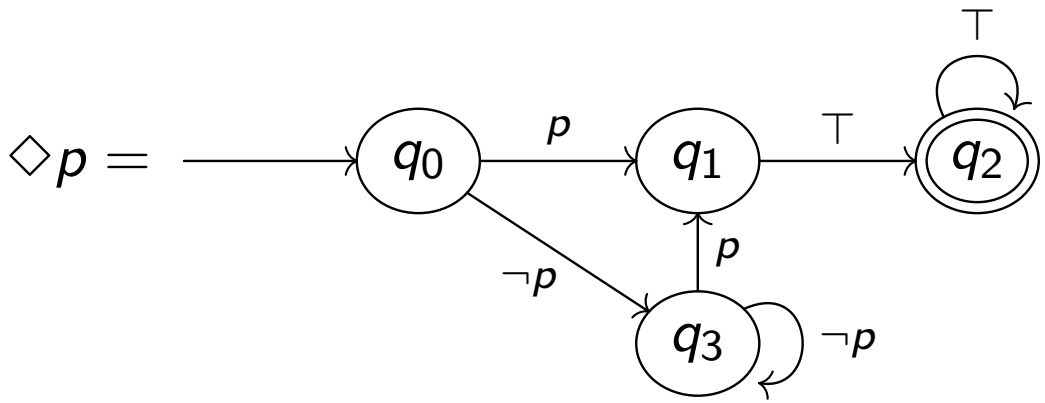
- Opérations faciles (polynomial en le nombre d'états) : $\cup$, $\cap$, concaténation
- Opération facile : tester le vide (aucun mot accepté)
- Opération coûteuse (exponentielle) : le complémentaire

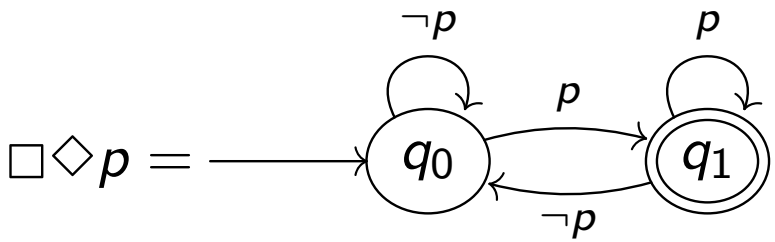
(Note : les automates de Büchi non déterministes sont strictement plus expressifs que les automates de Büchi déterministes. Par exemple : $\{a, b\}^* b^\omega$ n'est pas reconnaissable avec un automate déterministe.)

Transformation de φ en automate

On peut transformer toute formule LTL en un automate de Büchi reconnaissant le même langage (le même ensemble de traces).







Principe de la vérification LTL

Vérifier $\mathcal{M} \models \varphi$

1. Transformer $\mathcal{M}$ en un automate $B_{\mathcal{M}}$ (trivial)
2. Transformer φ en $B_{\neg\varphi}$
3. Construire $B_{\otimes}$ reconnaissant $L(B_{\mathcal{M}}) \cap L(B_{\neg\varphi})$
4. Tester si $L(B_{\otimes})$ est vide : si oui alors $M \models \varphi$; si non, on a un contre-exemple

Complexité en temps : $O(|M| \times 2^{|\varphi|}))$
Complexité en espace : **PSPACE**-complet

Équité

L'équité est une formule de LTL (p.e. $\Box\Diamond F$ ou $\Diamond\Box(\text{ENABLED } A) \Rightarrow \Box\Diamond A) \rightarrow$ directement pris en compte

Plan

TLA$^+$ Proof System (TLAPS)

TLAPS

- Un assistant de preuve pour TLA$^+$
- Écriture manuelle de la preuve selon une structure hiérarchique
- Chaque étape est mécaniquement vérifiée, soit directement, soit par des démonstrateurs externes :
  - SMT (logique du premier ordre, arithmétique)
  - Zenon (logique du premier ordre)
  - Isabelle (théorie des ensembles, induction, second ordre)
  - PTL (propositional temporal logic)
- Obtention d'un certificat formel vérifiable

Exemple : `xyplus1` (spécification)

MODULE *xyplus1*

EXTENDS *Naturals*, *FiniteSetTheorems*, *TLAPS*

VARIABLES x, y

$$\mathit{TypeInv} \triangleq (x \in \mathit{Nat} \land y \in \mathit{Nat})$$

$$\mathit{Ecart} \triangleq (0 \leq x - y \land x - y \leq 1)$$

$$\mathit{Init} \triangleq x = 0 \land y = 0$$

$$\mathit{Act1} \triangleq x' = y + 1 \land \text{UNCHANGED } \langle y \rangle$$

$$\mathit{Act2} \triangleq y' = x \land \text{UNCHANGED } \langle x \rangle$$

$$\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Act1} \lor \mathit{Act2}]_{\langle x, y \rangle}$$

Exemple : xyplus1 (preuves)

MODULE *xyplus1*

THEOREM $TypeInvOK \triangleq Spec \Rightarrow \Box TypeInv$
- $\langle 1 \rangle 1$ $Init \Rightarrow TypeInv$ BY DEF $Init$, $TypeInv$
- $\langle 1 \rangle 2$ $(Act1 \vee Act2) \wedge TypeInv \Rightarrow TypeInv'$ BY DEF $Act1$, $Act2$, $TypeInv$
- $\langle 1 \rangle 3$ UNCHANGED $\langle x, y \rangle \wedge TypeInv \Rightarrow TypeInv'$ BY DEF $TypeInv$
- $\langle 1 \rangle 4$ QED BY PTL, $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$ DEF $TypeInv$, $Spec$

THEOREM $EcartOK \triangleq Spec \Rightarrow \Box Ecart$
- $\langle 1 \rangle 1$ $Init \Rightarrow Ecart$ BY DEF $Init$, $Ecart$
- $\langle 1 \rangle 2$ $(Act1 \vee Act2) \wedge TypeInv \wedge Ecart \Rightarrow Ecart'$
  - $\langle 2 \rangle 1$ $Act1 \wedge TypeInv \wedge Ecart \Rightarrow Ecart'$ BY DEF $Act1$, $TypeInv$, $Ecart$
  - $\langle 2 \rangle 2$ $Act2 \wedge TypeInv \wedge Ecart \Rightarrow Ecart'$ BY DEF $Act2$, $TypeInv$, $Ecart$
  - $\langle 2 \rangle 3$ QED BY $\langle 2 \rangle 2$, $\langle 2 \rangle 1$
- $\langle 1 \rangle 3$ UNCHANGED $\langle x, y \rangle \wedge Ecart \Rightarrow Ecart'$ BY DEF $Ecart$
- $\langle 1 \rangle 4$ QED BY PTL, $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $TypeInvOK$ DEF $Ecart$, $Spec$, $TypeInv$

THEOREM $Spec \Rightarrow \Box(TypeInv \wedge Ecart)$ BY $TypeInvOK$, $EcartOK$

Conclusion

Vérification par preuve

Preuve manuelle assistée :

- Expertise nécessaire à la fois dans le domaine et le vérificateur
- Système infini, de grande taille, paramétré
- De gros progrès récents avec la décharge partielle sur des vérificateurs externes automatiques

Vérification de modèles

- Vérification automatique
- Systèmes finis de petite taille (quelques millions d'états)
- Nécessite la construction de l'espace d'état
- LTL : $O(|M| \times 2^{|\varphi|})$)
- CTL : $O(|M| \times |\varphi|)$, mais l'équité complexifie les formules

Pour aller plus loin

- *Symbolic model checking* : représentation d'un ensemble d'états par une formule
  Intéressant quand la spécification est elle-même décrite par des formules (cas de TLA$^+$)
- Model checking à la volée : vérification au fur et à mesure de la construction de l'espace d'état, permet de trouver plus vite un contre-exemple s'il existe.
- *Bounded model checking* : borner la profondeur explorée.
  Efficace à faible profondeur (dépliage du modèle sous la forme d'une formule SAT) mais ne garantit pas la correction totale.
- Collaboration vérification par preuve $\leftrightarrow$ vérification de modèles