Vérification par abstraction : état de l'art et quelques perspectives

Grégoire Sutre

LaBRI (Bordeaux)

Vérification par abstraction : ~~état de l'art~~ et quelques perspectives

Grégoire Sutre

LaBRI (Bordeaux)

Vérification par abstraction : quelques éléments et quelques perspectives

Grégoire Sutre

LaBRI (Bordeaux)

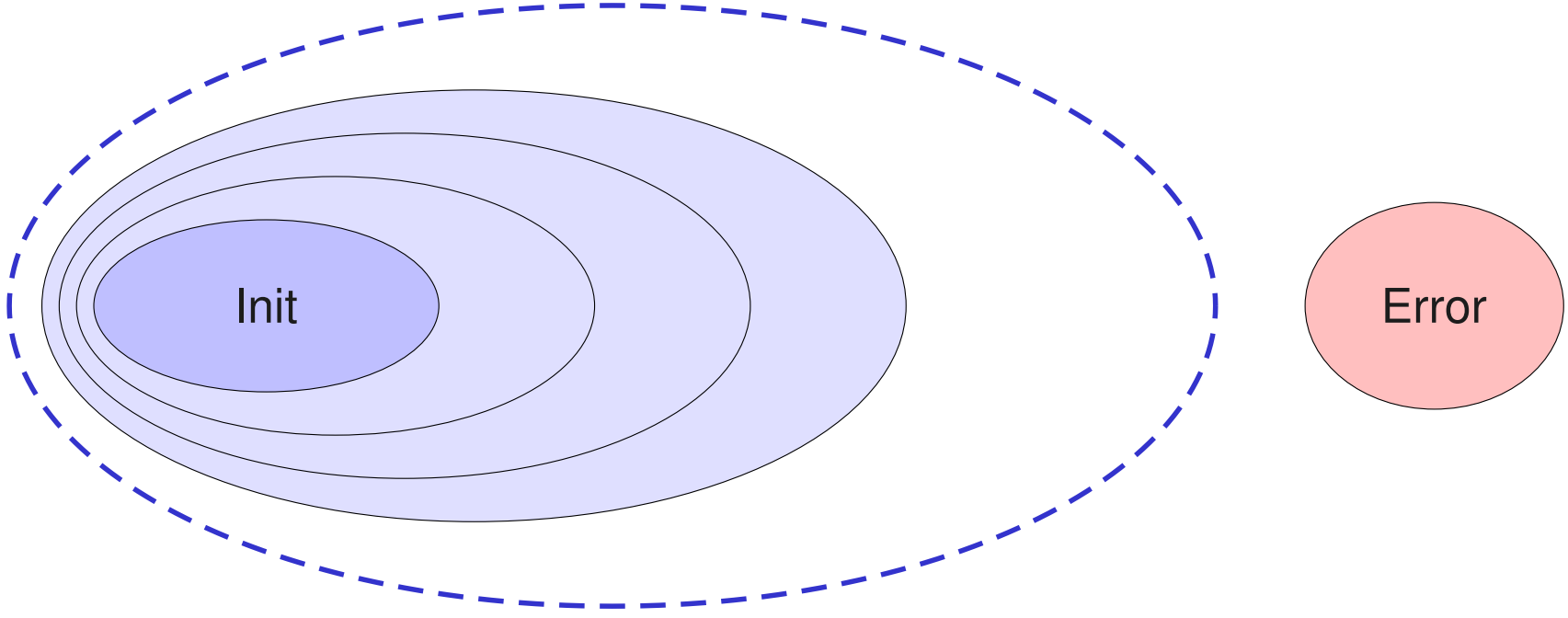
Outline

1. Introduction
2. Abstraction
3. Interprétation abstraite
4. Abstraction par prédicats
5. Raffinement guidé par contre-exemple
6. Perspectives

Vérification de $\mathrm{AG}\,\neg Error$

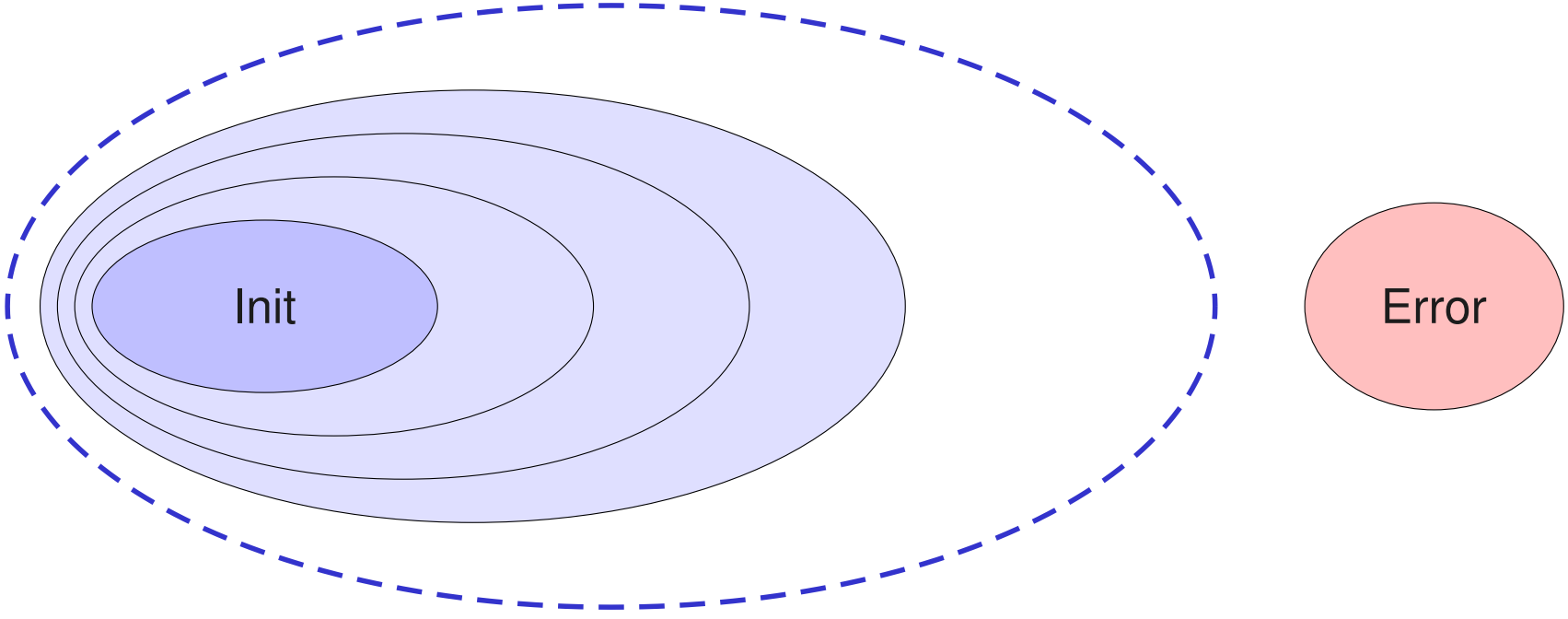
Vérification de $\mathrm{AG}\,\neg Error$

Vérification de $\mathrm{AG}\,\neg Error$



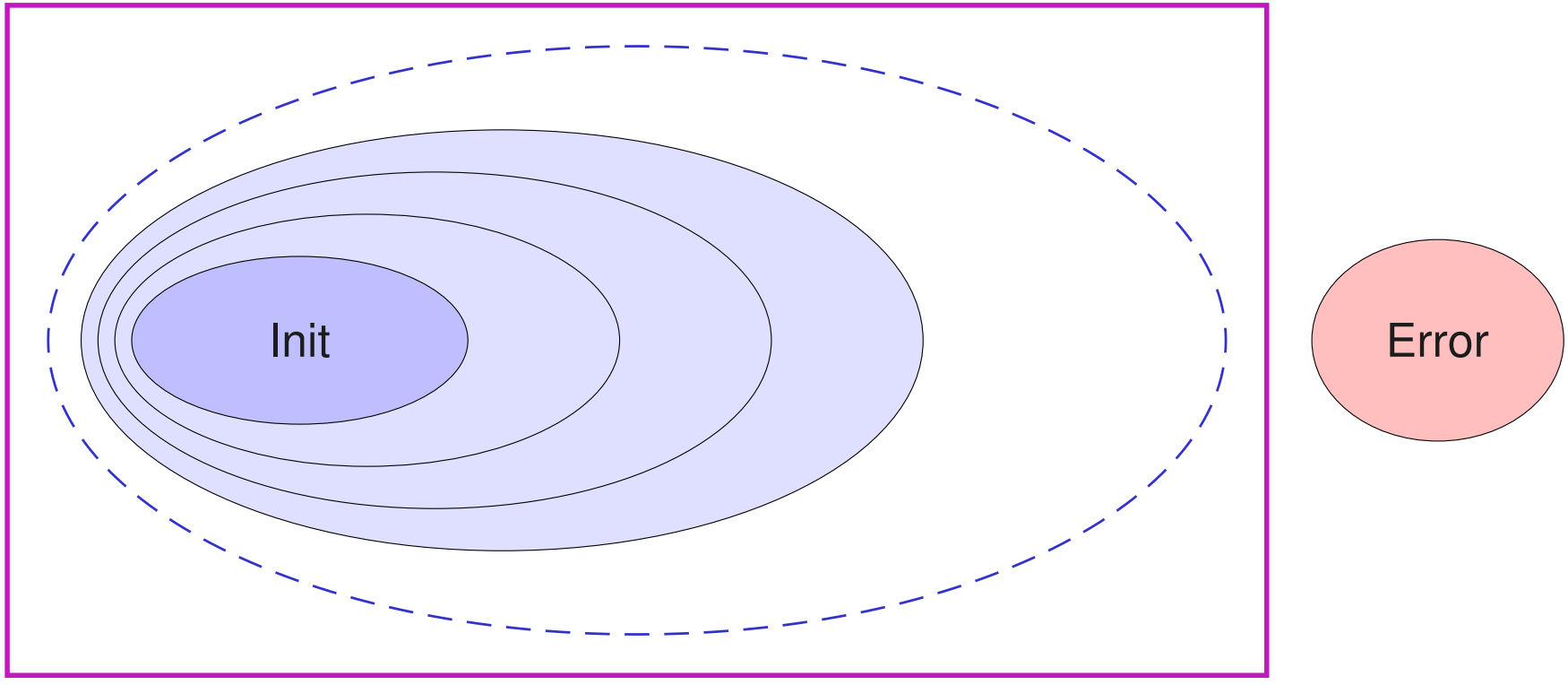
- l'itération ne termine pas (trop d'états)

Vérification de AG $\neg Error$



- l'itération ne termine pas (trop d'états)
- Solutions:
  - accélération: calcul exact, mais trop de détails inutiles

Vérification de AG $\neg Error$



- l'itération ne termine pas (trop d'états)
- Solutions:
  - accélération: calcul exact, mais trop de détails inutiles
  - abstraction: pour se débarasser de ces détails superflus

Motivations

- Réduction de la complexité du modèle à vérifier
  - atténuer l'explosion du nombre d'états
  - vérifier des systèmes infinis en ré-utilisant des model-checkers pour systèmes finis
- Elimination des détails superflus vis-à-vis de la propriété à vérifier
  - crucial pour la vérification des systèmes réels (circuits, logiciel, ...)
- Implantation de spécifications (méthode B)

Principe général

$$\mathcal{A} \models \varphi \implies \mathcal{T} \models \varphi$$

Mise en pratique :

- Construction d'abstractions (abstraction par prédicats)
- Validation des contre-exemples abstraits
- Raffinement de l'abstraction guidé par (faux) contre-exemples abstraits
- Automatisation

Outline

1. Introduction
2. Abstraction
3. Interprétation abstraite
4. Abstraction par prédicats
5. Raffinement guidé par contre-exemple
6. Perspectives

Simulation

- $\mathcal{T}_1 = (S_1, R_1)$ et $\mathcal{T}_2 = (S_2, R_2)$: systèmes de transitions
- Relation entre états : $\rho \subseteq S_1 \times S_2$

Définition. $R_1 \sqsubseteq_\rho R_2$ et $\mathcal{T}_1 \sqsubseteq_\rho \mathcal{T}_2$ si $\rho^{-1} R_1 \subseteq R_2 \rho^{-1}$

- $\mathcal{T}_1 \sqsubseteq_\rho \mathcal{T}_2$ se lit :
  - $\mathcal{T}_1$ ρ–simule $\mathcal{T}_2$
  - $\mathcal{T}_2$ est une ρ–abstraction de $\mathcal{T}_1$

Préservation de propriétés universelles

- $\mathcal{T} = (S, R)$ et $\mathcal{A} = (A, R_\alpha)$: systèmes de transitions
- $I \subseteq S$ et $I_\alpha \subseteq A$: états initiaux
- $Lit = Prop \cup \{\neg p | p \in Prop\}$
- littéraux interprétés par des ensembles d'états de $\mathcal{T}$ et de $\mathcal{A}$

Si :

- $\mathcal{T} \sqsubseteq_\rho \mathcal{A}$ et $I \subseteq \rho^{-1} I_\alpha$
- pour tout $(s, a) \in \rho$ et $p \in Lit$, on a : $a \models p \implies s \models p$

Alors : pour toute formule $\varphi \in \forall \mathrm{CTL}^*$, on a : $\mathcal{A}, I_\alpha \models \varphi \implies \mathcal{T}, I \models \varphi$

Eléments de complétude

- Cette approche est complète pour les formules d'invariance
- Si $\mathcal{T}, I \models \mathrm{AG}\, p$ alors il suffit de choisir :
  - l'abstraction $\mathcal{A} = (\{a\}, \{(a, a)\})$, avec $a = post^*(I)$
  - $I_\alpha = \{a\}$
  - $a \models p$
- Ce n'est plus le cas pour la formule $\mathrm{AGF}\, p$
- La vérification par abstraction étendue [KPV01] permet de garantir la complétude (pour $\forall$ Büchi)

Préservation de propriétés générales

- $\mathcal{A} = (A, R_\alpha^\Box, R_\alpha^\Diamond)$: 2 relations de transitions
- Interprétation des formules de logique temporelle :
  - $R_\alpha^\Box$ pour les quantification universelles (A de CTL^*)
  - $R_\alpha^\Diamond$ pour les quantification existentielles (E de CTL^*)

Si :

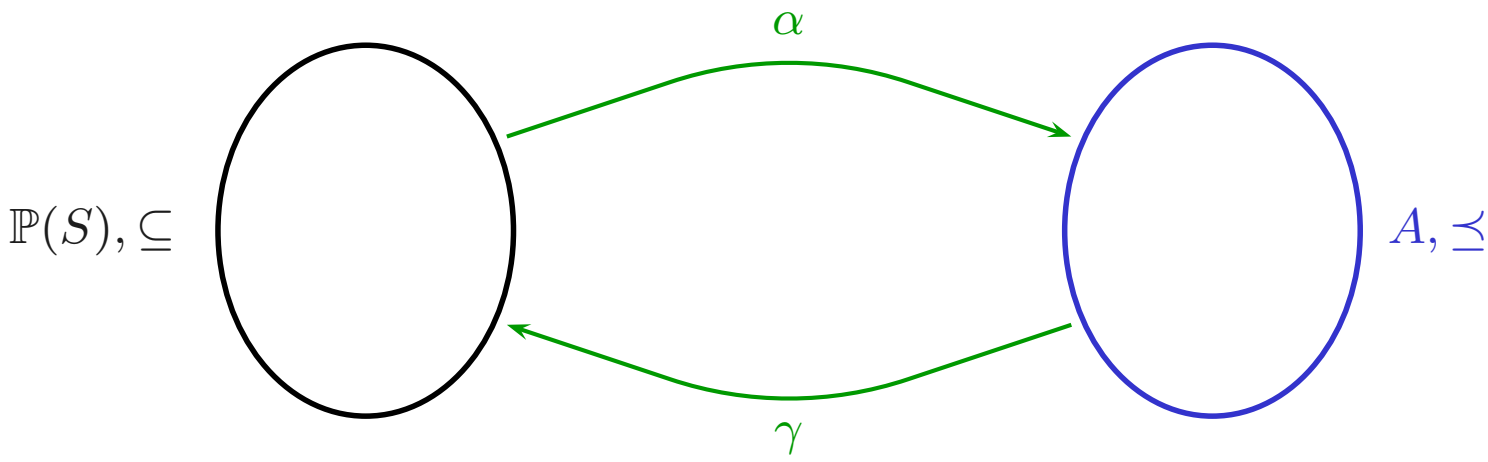
- $R \sqsubseteq_\rho R_\alpha^\Box$ et $R_\alpha^\Diamond \sqsubseteq_\rho R$ et $I \subseteq \rho^{-1} I_\alpha$
- pour tout $(s, a) \in \rho$ et $p \in Lit$, on a : $a \models p \implies s \models p$

Alors : pour toute formule $\varphi \in \mathrm{CTL}^*$, on a : $\mathcal{A}, I_\alpha \models \varphi \implies \mathcal{T}, I \models \varphi$

Outline

1. Introduction
2. Abstraction
3. Interprétation abstraite
4. Abstraction par prédicats
5. Raffinement guidé par contre-exemple
6. Perspectives

Connection de Galois



- fonctions totales et monotones de concrétisation γ et d'abstraction α
- $\gamma \circ \alpha$ fermeture supérieure et $\alpha \circ \gamma$ fermeture inférieure
- la donnée de l'une détermine l'autre : $\alpha(X) = \bigwedge\{a \in A \mid X \subseteq \gamma(a)\}$
- si de plus $a \preceq a' \iff \gamma(a) \subseteq \gamma(a')$ alors : $\alpha \circ \gamma = Id$

Système de transitions abstrait

- $\mathcal{T} = (S, R)$: système de transition (concret)
- $(A, \preceq)$: domaine abstrait partiellement ordonné (treillis)
- On définit $\mathcal{A} = (A, R_\alpha)$ avec $(a, a') \in R_\alpha$ si $a' = \alpha(post(\gamma(a)))$
- $\mathcal{A}$ est une ρ–abstraction de $\mathcal{T}$, avec $\rho = \{(s, a) | s \in \gamma(a)\}$
- On définit I_α par $I_\alpha = \{\alpha(s_0) | s_0 \in I\}$
- On pose $a \models p$ dès que $\gamma(a) \models p$ (i.e. $s \models p$ pour tout $s \in \gamma(a)$)

Ainsi : pour toute formule $\varphi \in \forall\,\mathrm{CTL}^*$, on a : $\mathcal{A}, I_\alpha \models \varphi \implies \mathcal{T}, I \models \varphi$

Système de transitions abstrait mixte

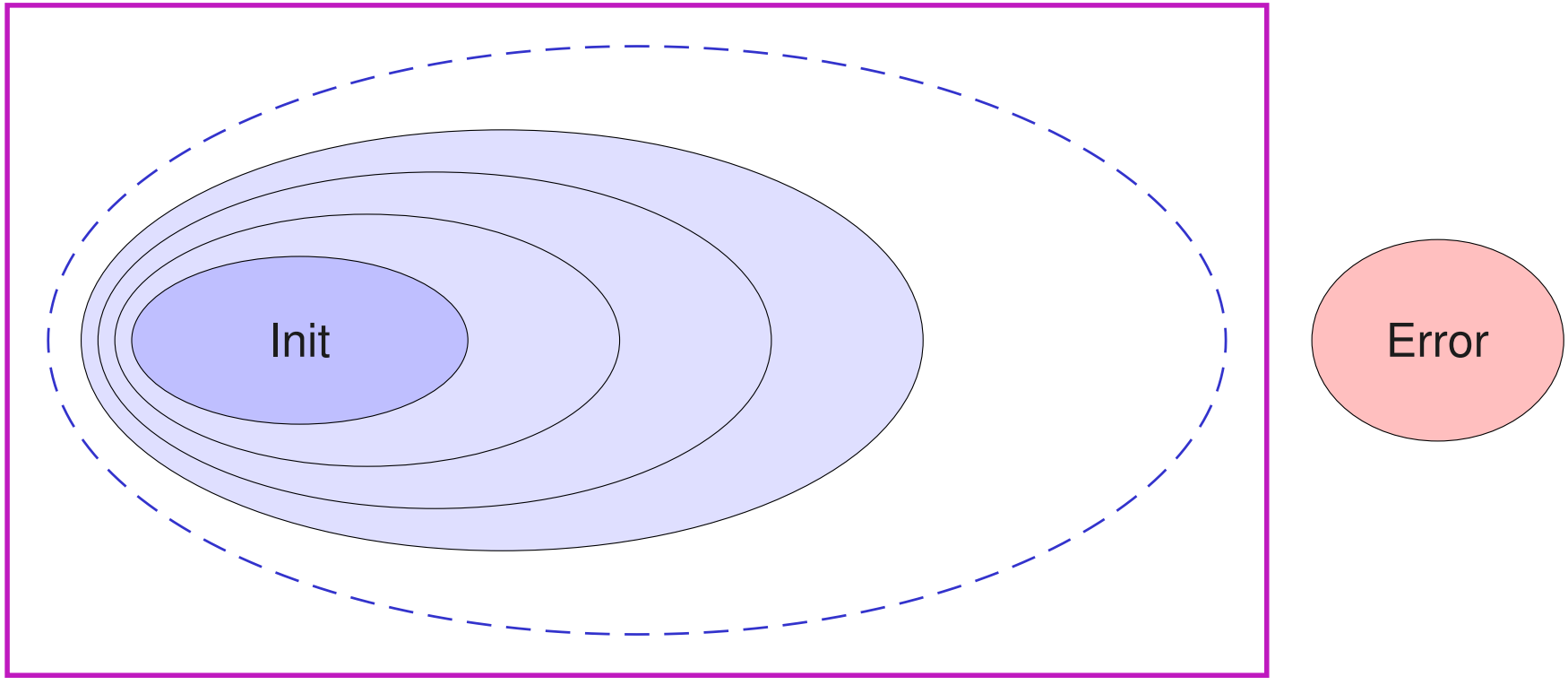
- On définit $\mathcal{A} = (A, R_\alpha^\Box, R_\alpha^\Diamond)$ avec :
  - $aR_\alpha^\Box a'$ si $a' = \alpha(post(\gamma(a)))$
  - $aR_\alpha^\Diamond a'$ s'il existe $X \subseteq S$ tel que $a' = \alpha(X)$ et $pre(X) \supseteq \gamma(a)$
  - on choisit les plus petits X (si possible)

Ainsi : pour toute formule $\varphi \in \mathrm{CTL}^*$, on a : $\mathcal{A}, I_\alpha \models \varphi \implies \mathcal{T}, I \models \varphi$

Outline

1. Introduction
2. Abstraction
3. Interprétation abstraite
4. **Abstraction par prédicats**
5. Raffinement guidé par contre-exemple
6. Perspectives

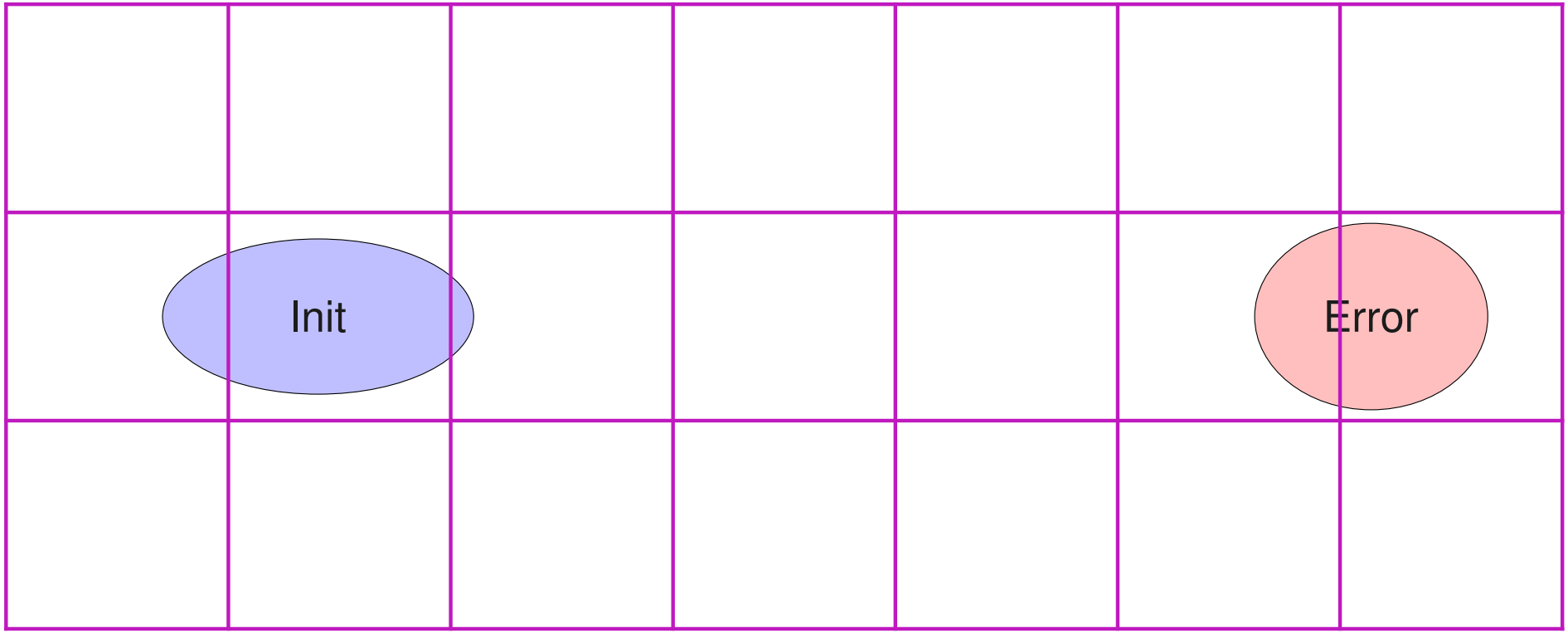
Vérification de AG $\neg Error$



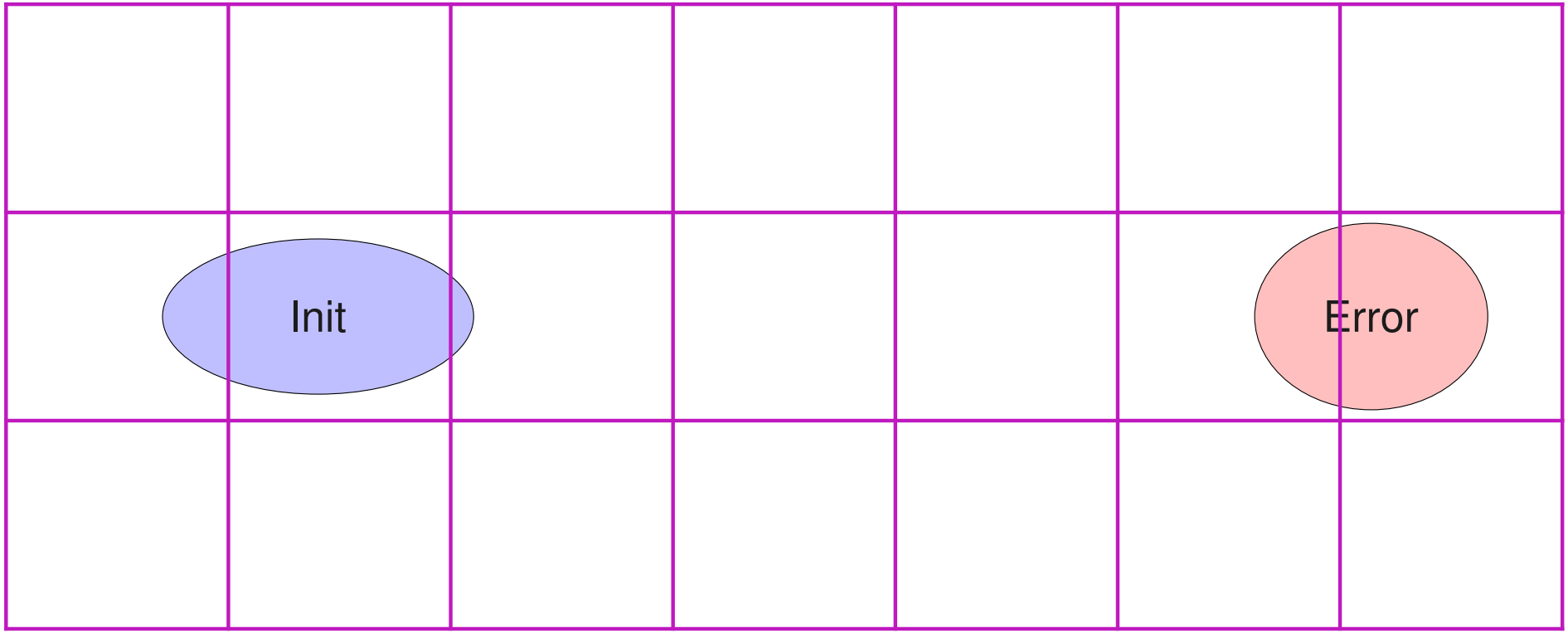
- l'itération ne termine pas (trop d'états)
- Solutions:
  - accélération: calcul exact, mais trop de détails inutiles
  - abstraction: pour se débarasser de ces détails superflus

Abstraction par prédicats [GS97]

Abstraction par prédicats [GS97]

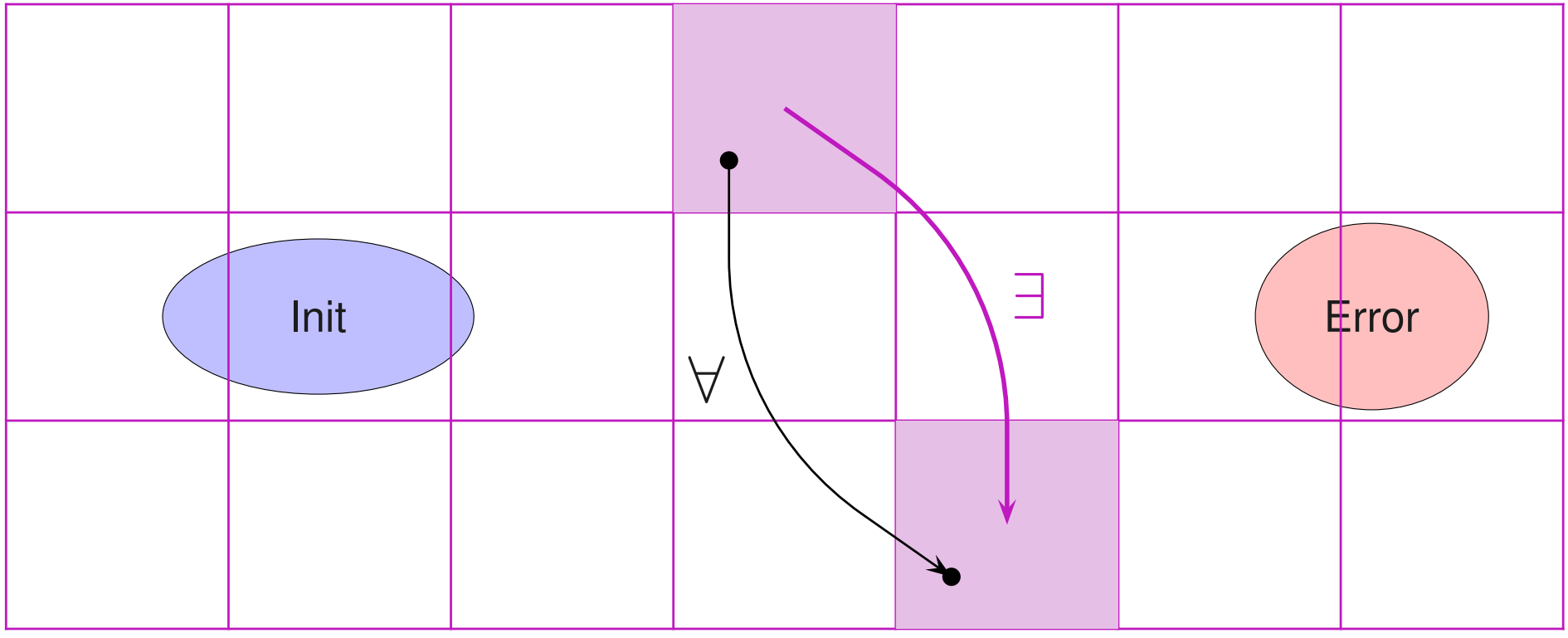


Abstraction par prédicats [GS97]



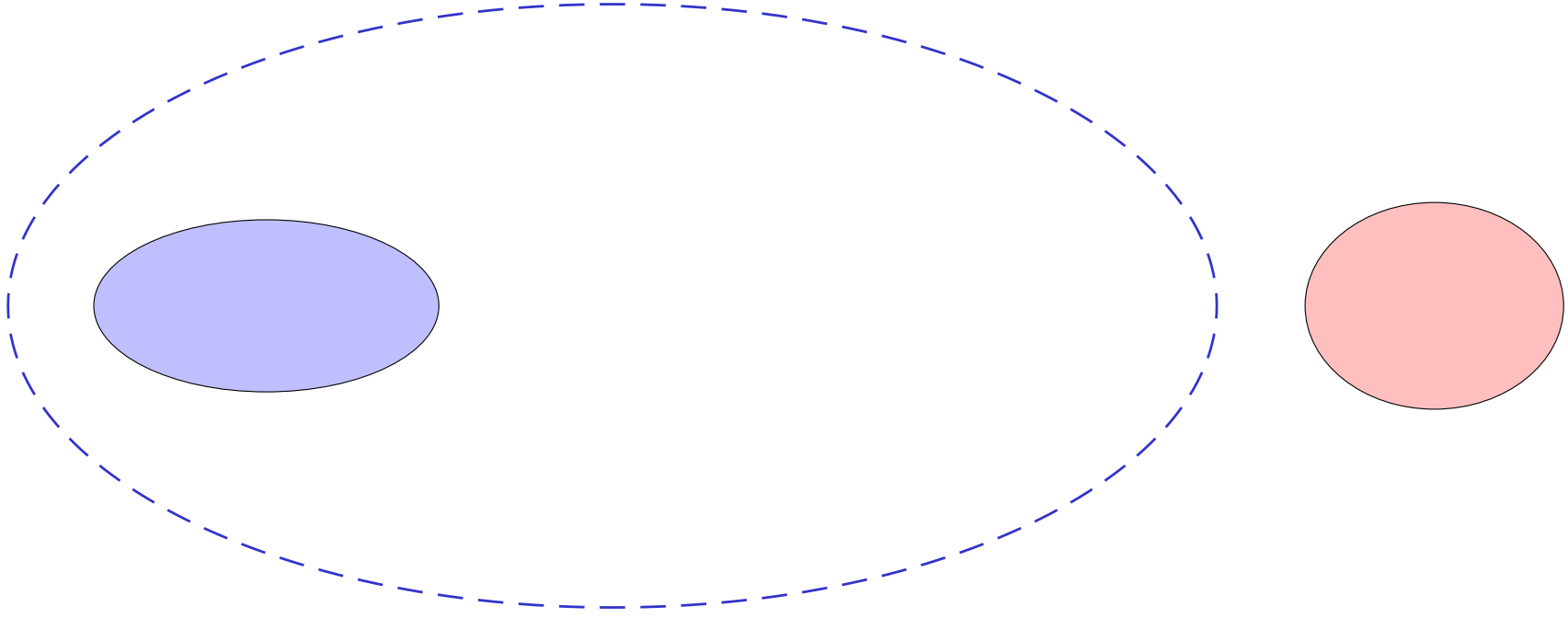
- $P_1, P_2, \ldots, P_n$: predicats ($[\![P_i]\!]$: sous-ensemble de l'espace d'états)
- Etats abstraits (boîtes) : valuations $\{P_1, P_2, \ldots, P_n\} \rightarrow \{\texttt{true}, \texttt{false}\}$

Abstraction par prédicats [GS97]

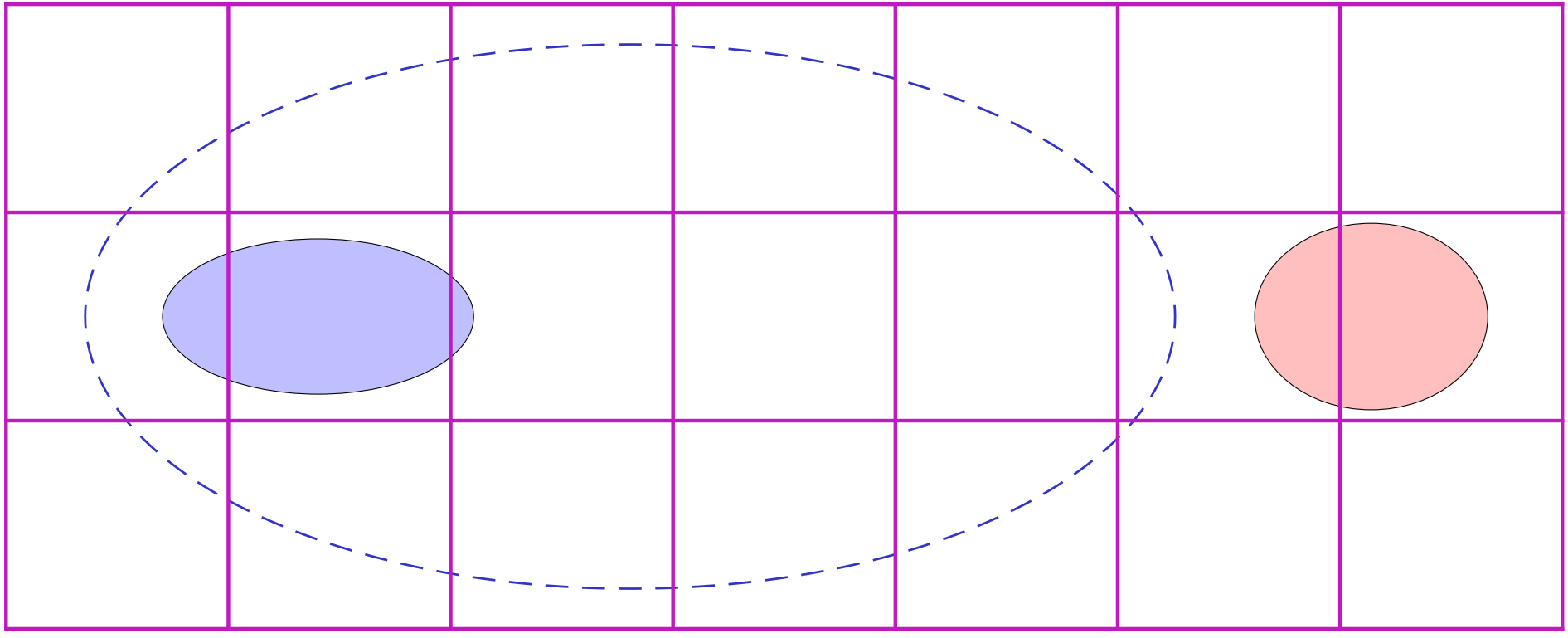


- $P_1, P_2, \ldots, P_n$: predicats ($[\![P_i]\!]$: sous-ensemble de l'espace d'états)
- Etats abstraits (boîtes) : valuations $\{P_1, P_2, \ldots, P_n\} \rightarrow \{\texttt{true}, \texttt{false}\}$
- Abstraction conservative (calculée à l'aide de procédures de décision)

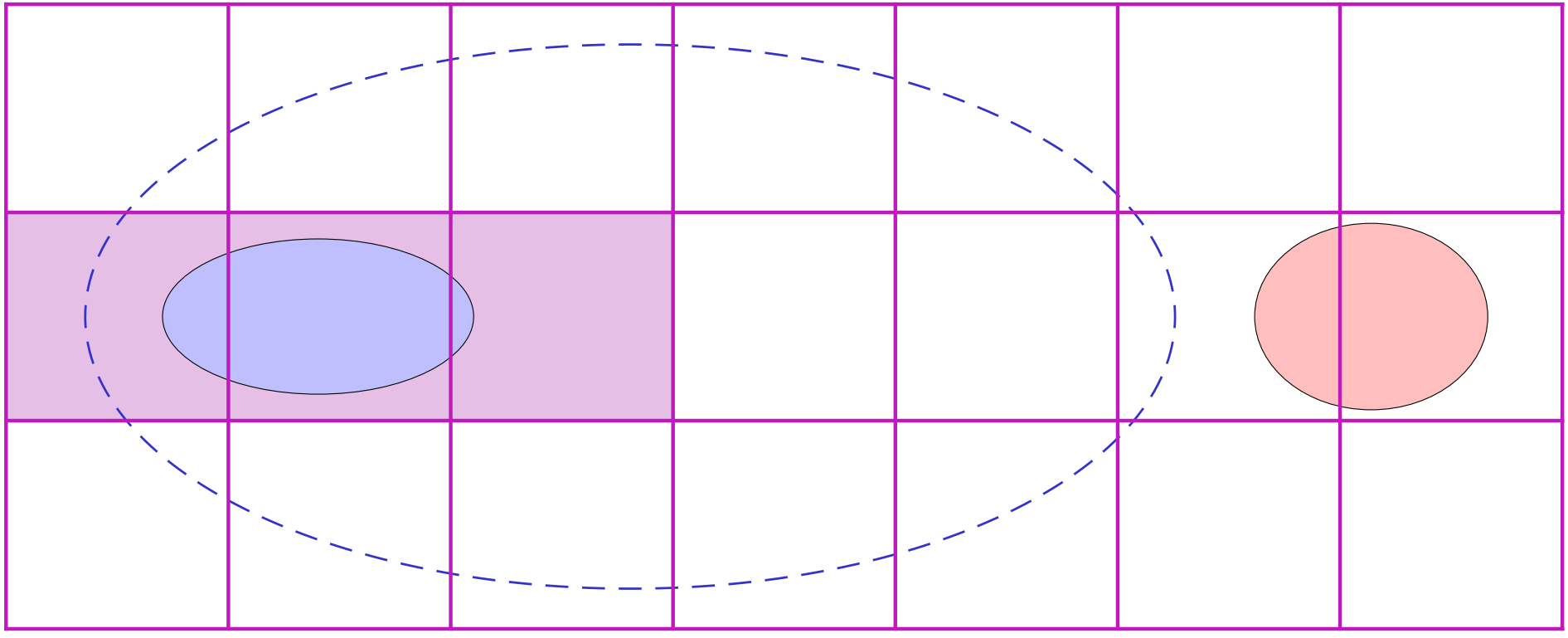
Vérification de l'abstraction par prédicats



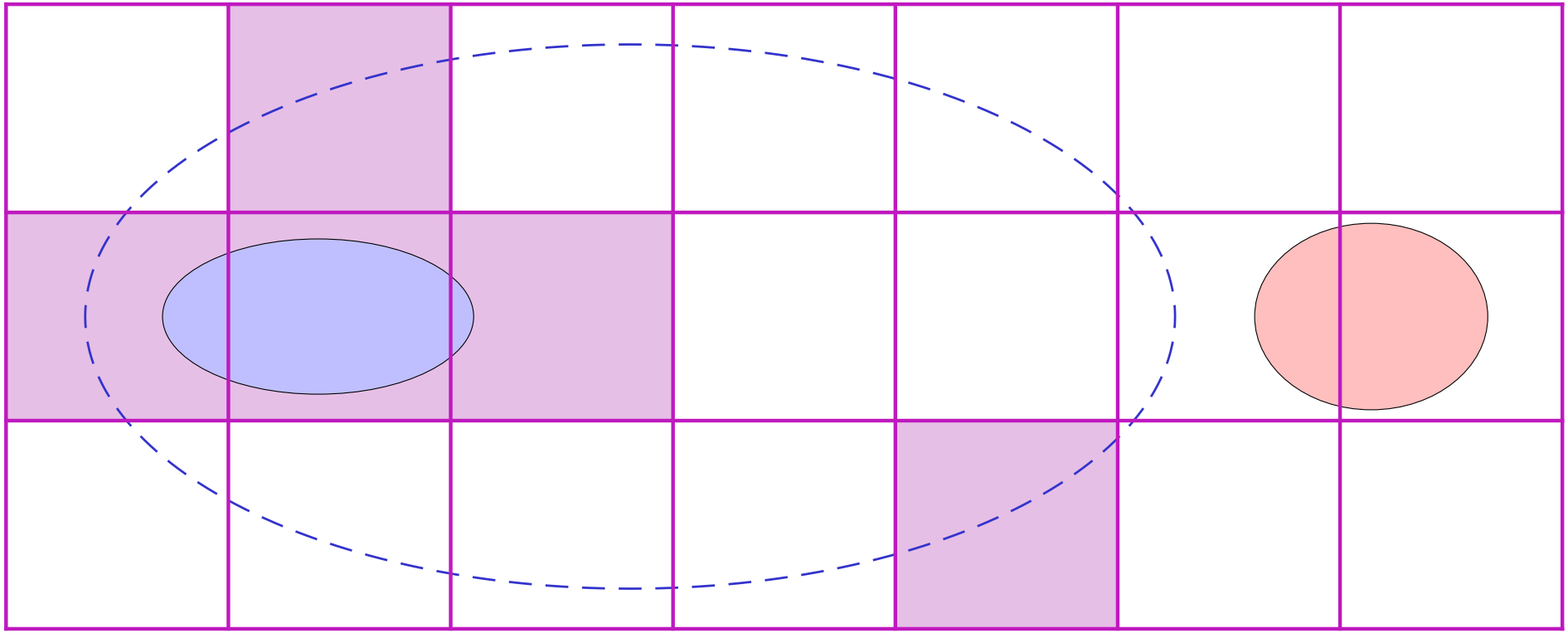
Vérification de l'abstraction par prédicats



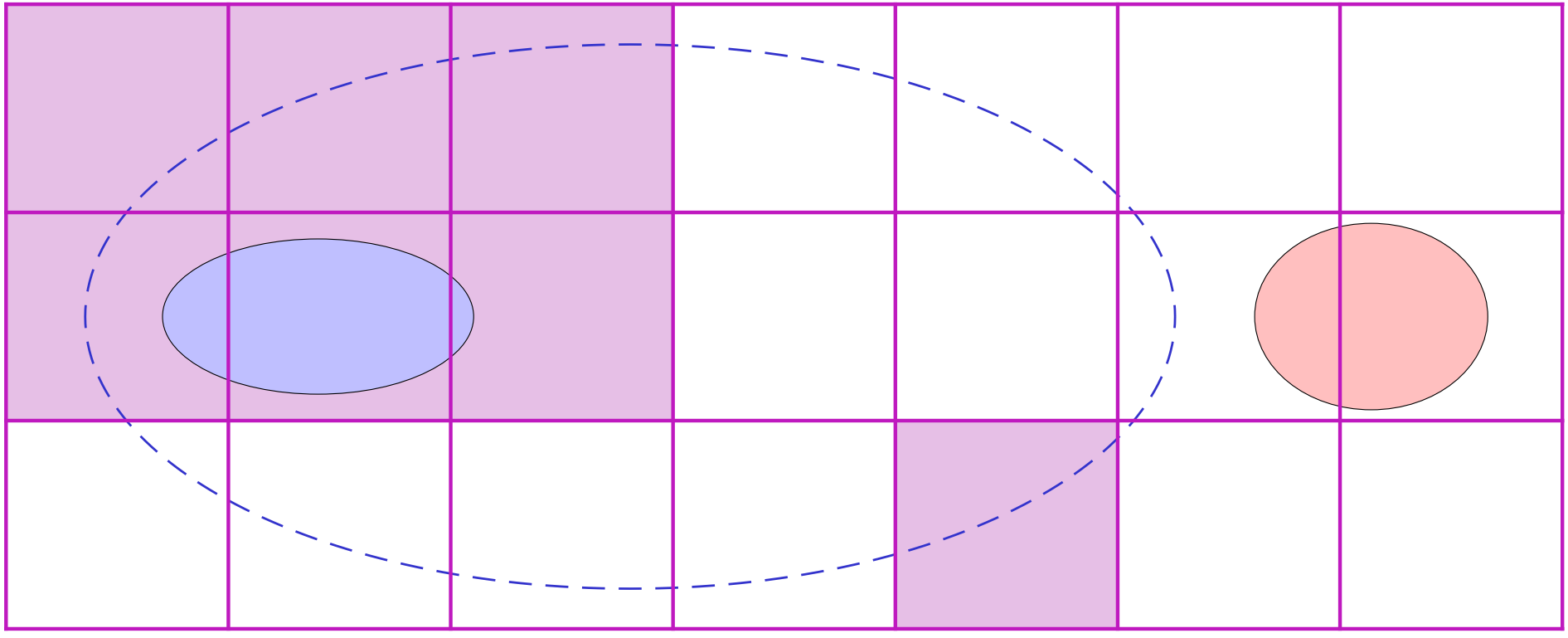
Vérification de l'abstraction par prédicats



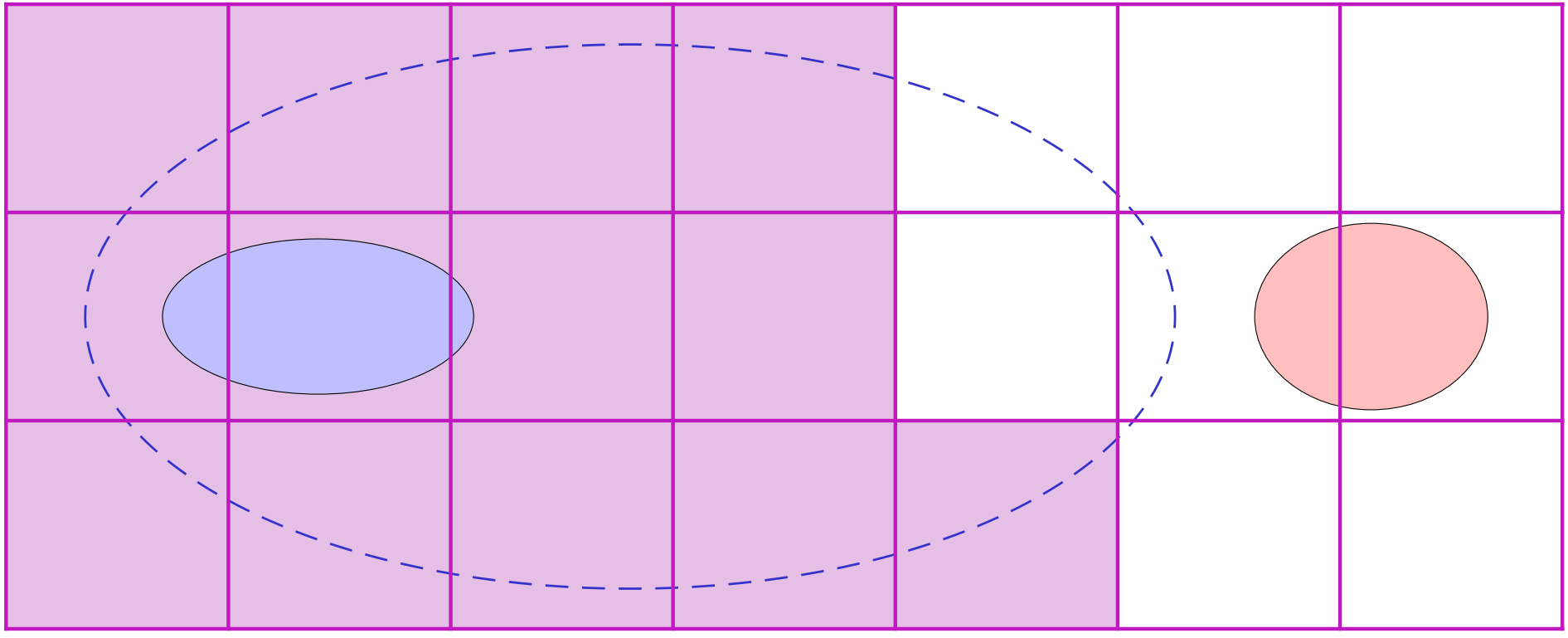
Vérification de l'abstraction par prédicats



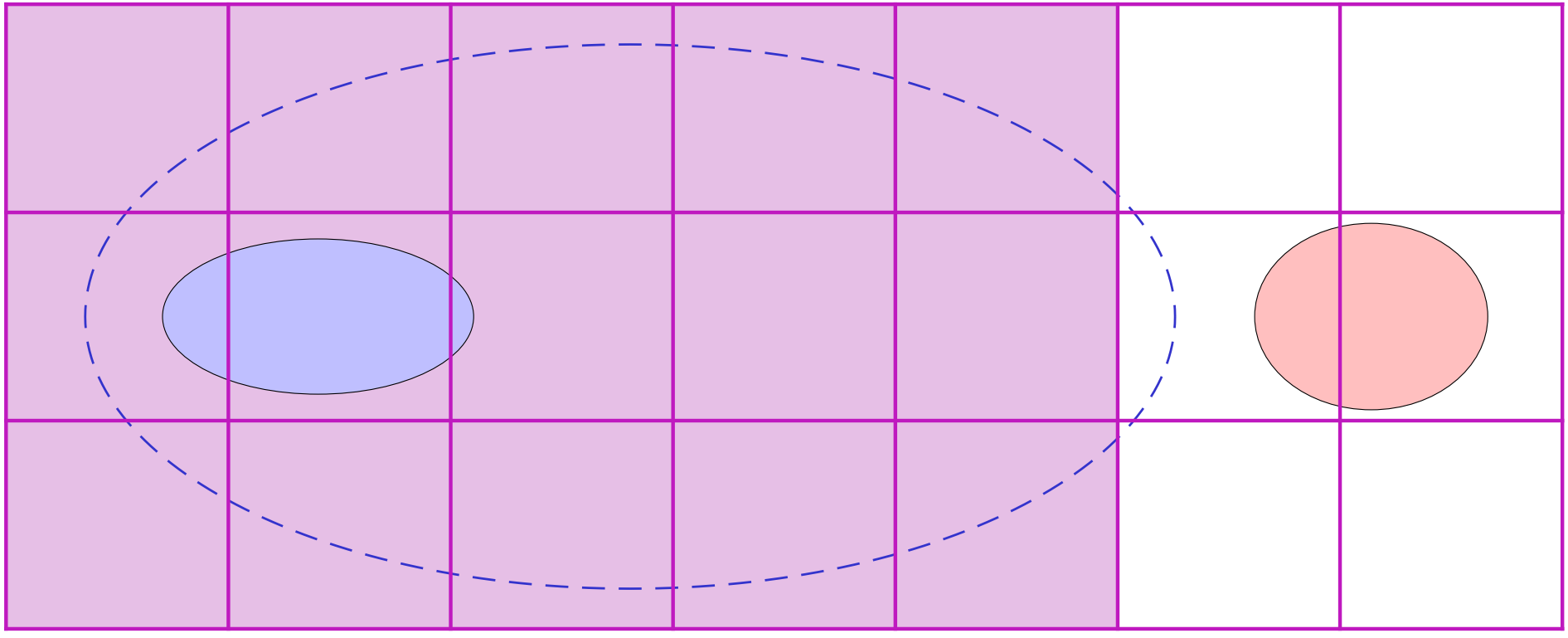
Vérification de l'abstraction par prédicats



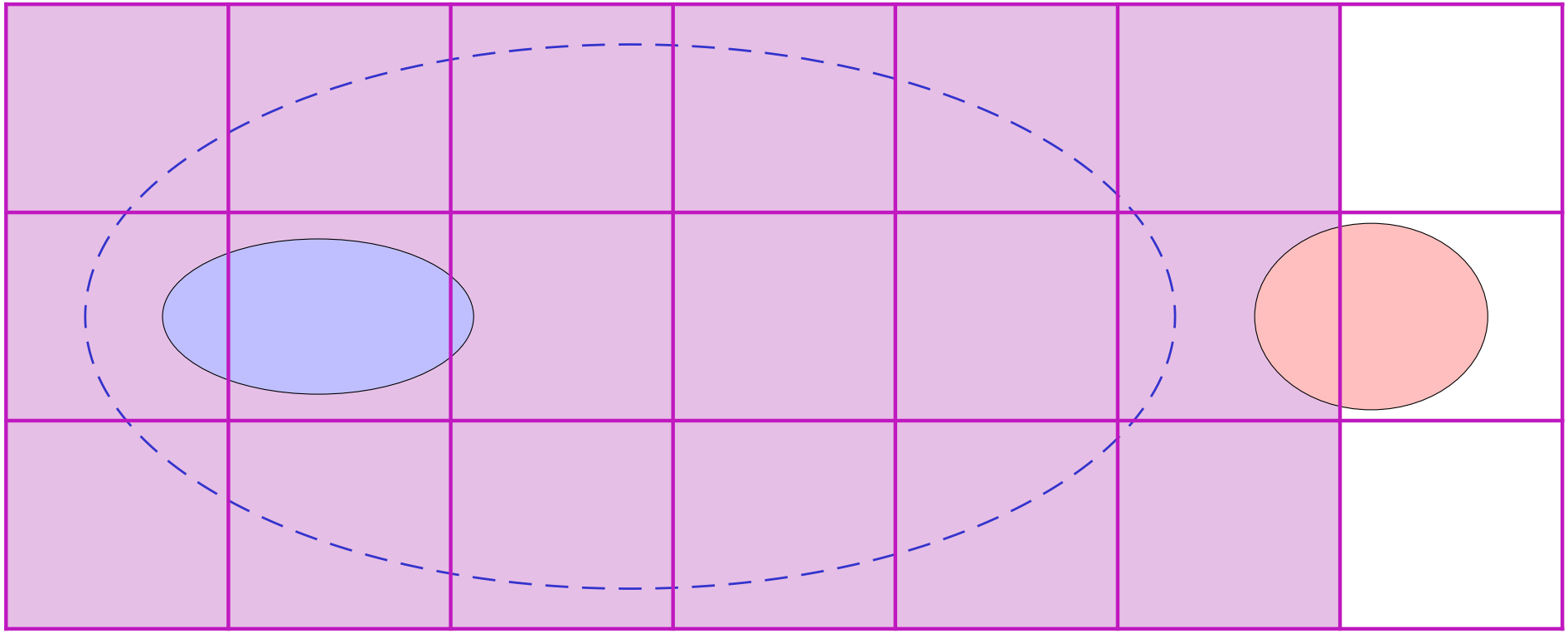
Vérification de l'abstraction par prédicats



Vérification de l'abstraction par prédicats

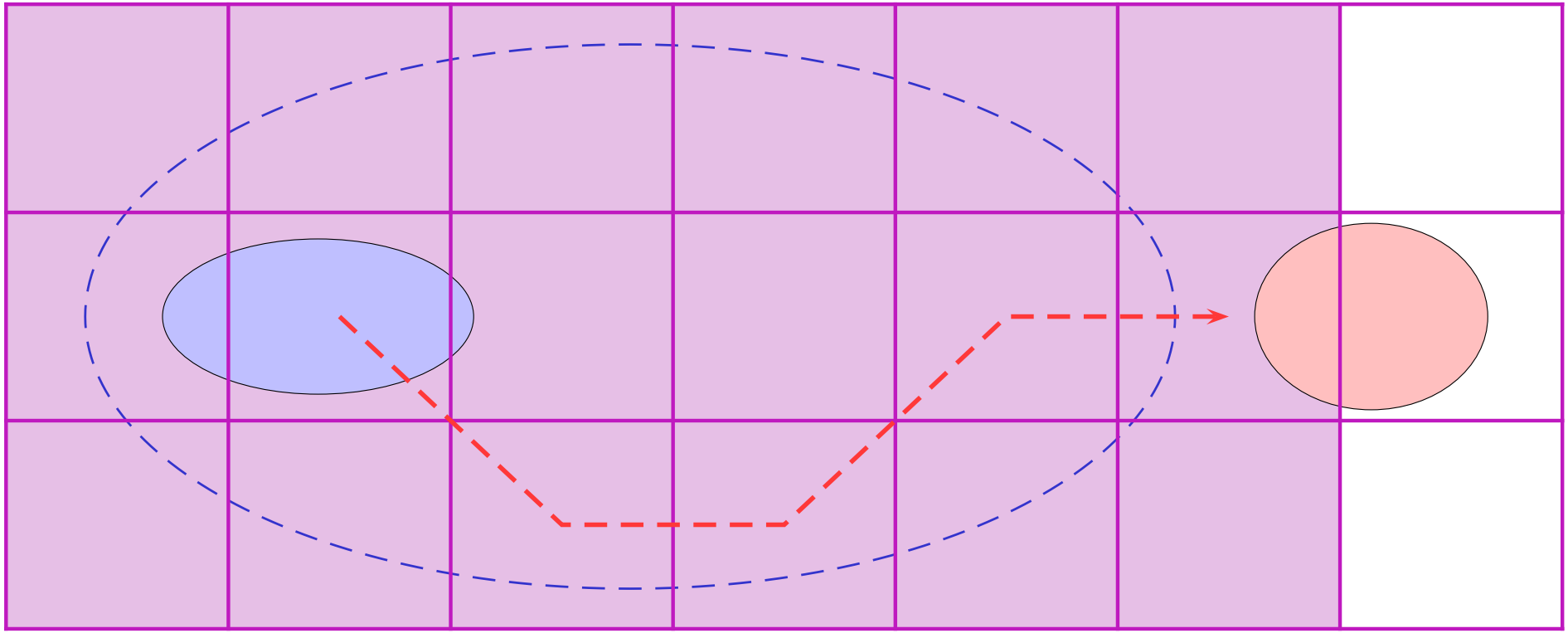


Vérification de l'abstraction par prédicats



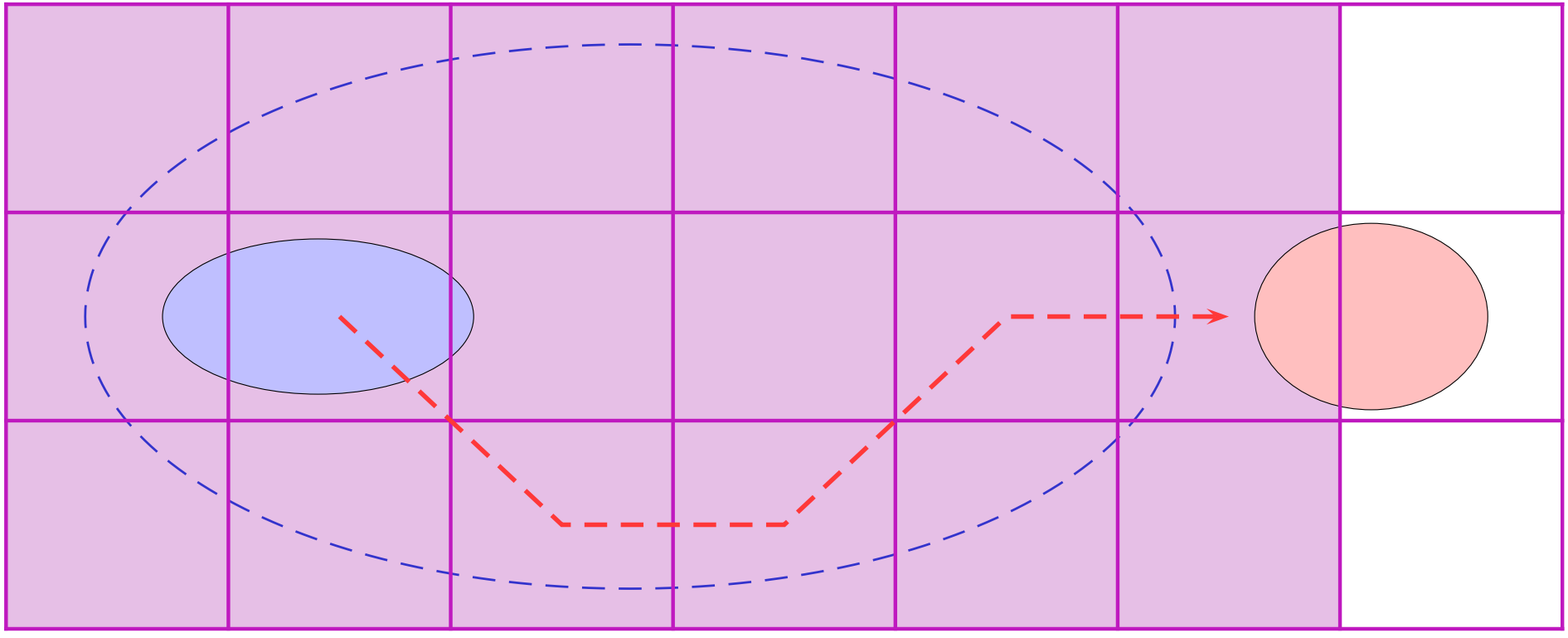
- Abstraction trop grossière

Vérification de l'abstraction par prédicats



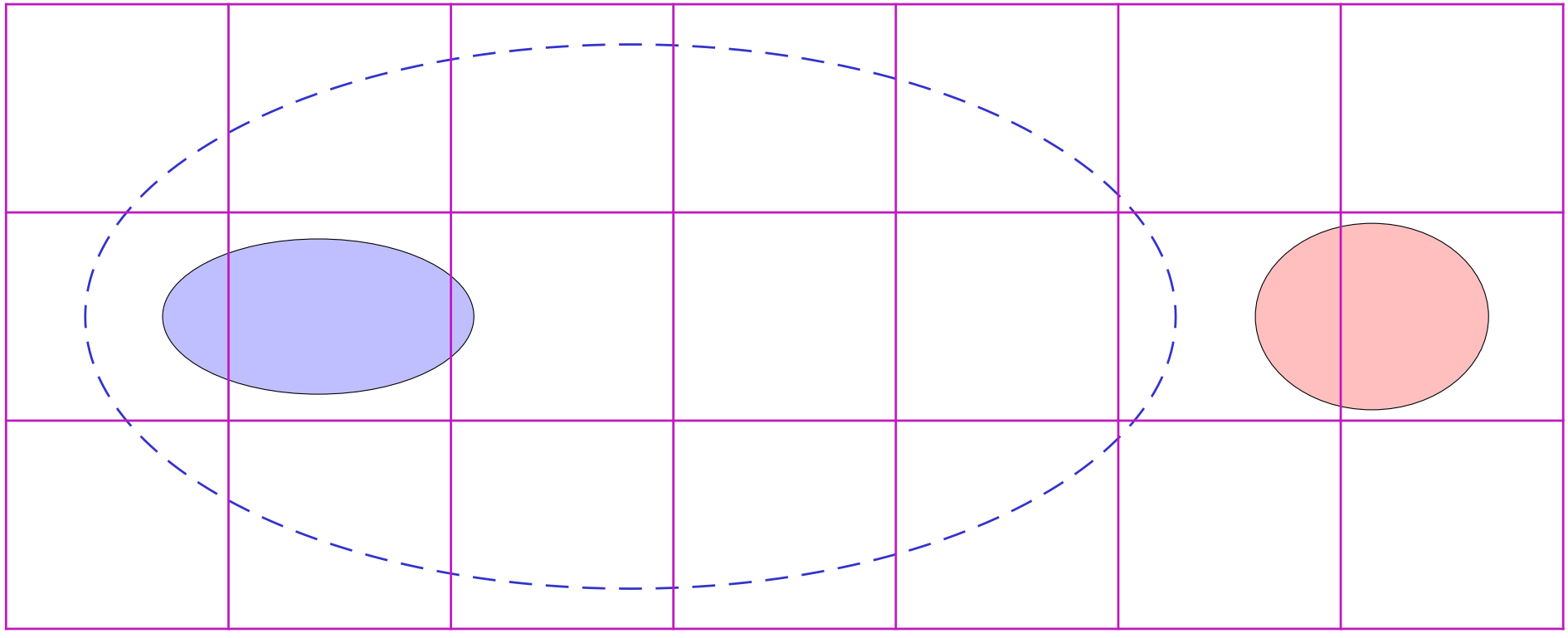
- Abstraction trop grossière

Vérification de l'abstraction par prédicats

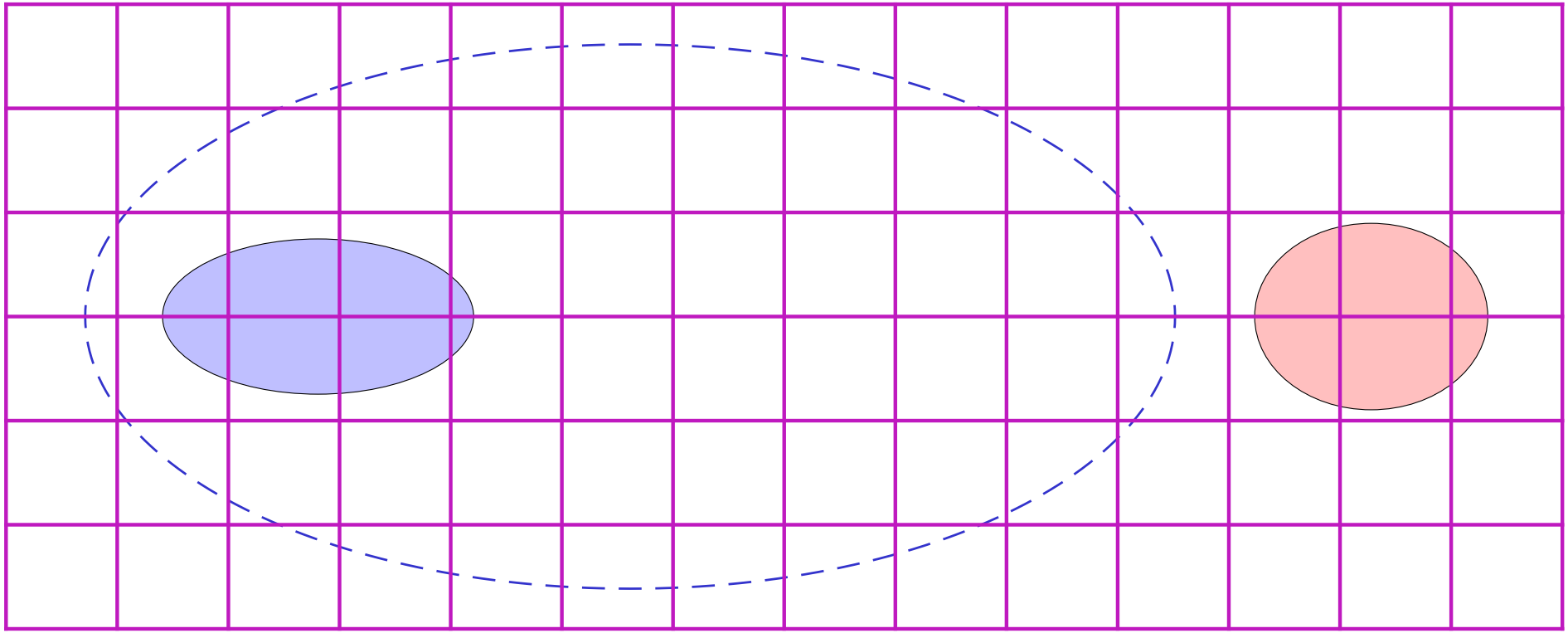


- Abstraction trop grossière
- Raffinement nécessaire pour éviter ce contre-exemple erroné
  - découpage des boîtes en sous-boîtes : ajout de nouveaux prédicats

Raffinement de l'abstraction par prédicats

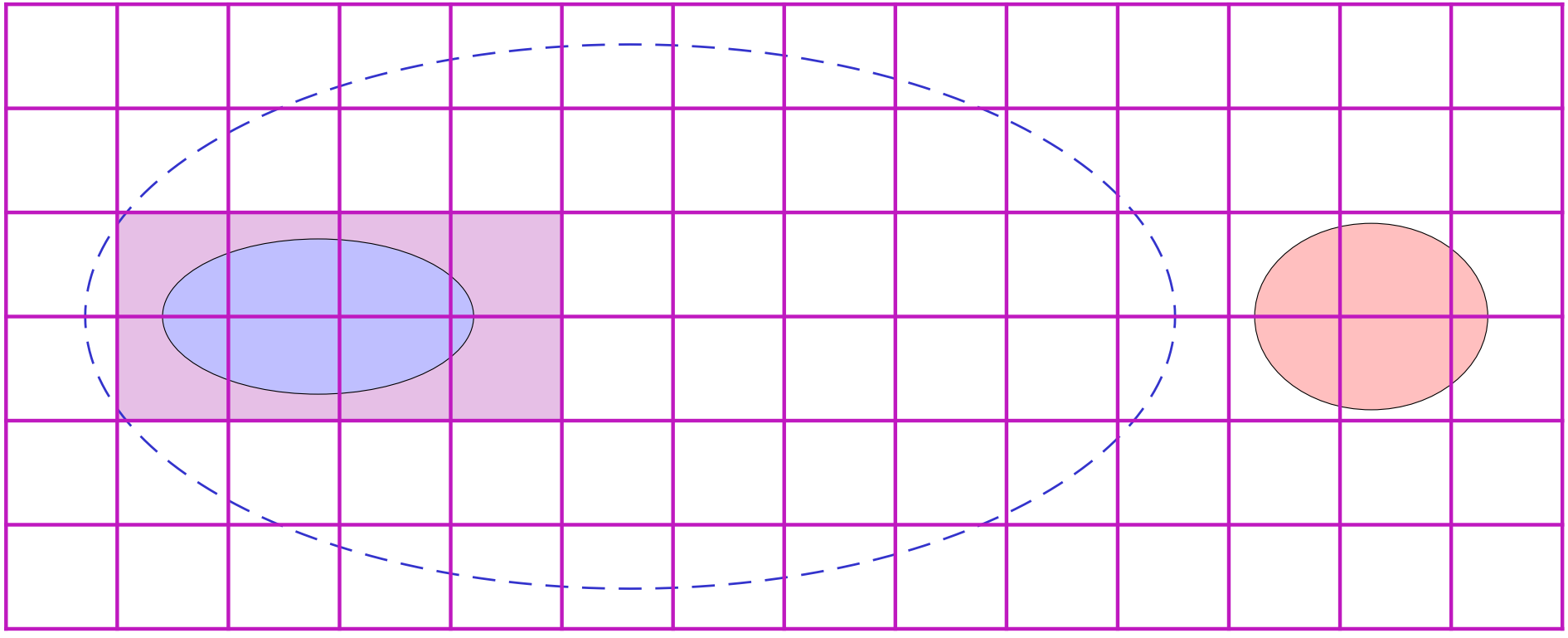


Raffinement de l'abstraction par prédicats



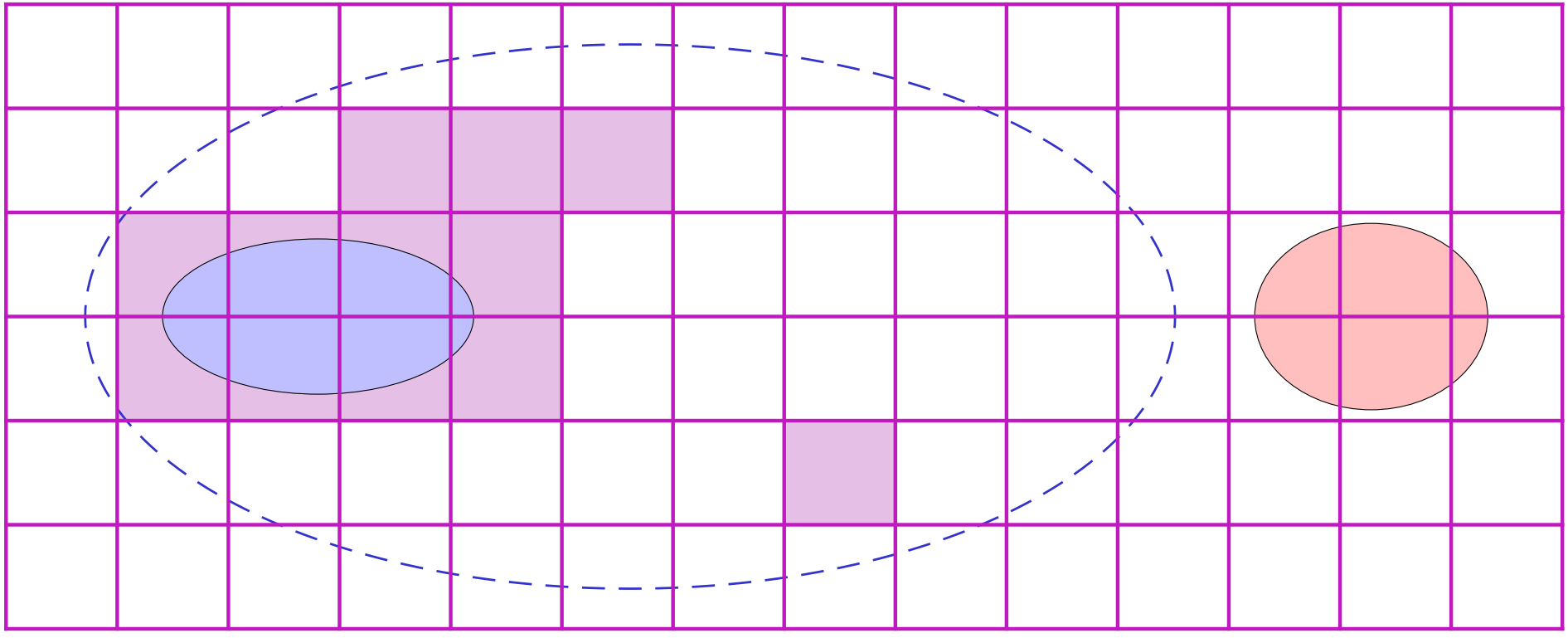
- Raffinement: ajout de nouveaux prédicats
- Grille plus fine

Raffinement de l'abstraction par prédicats



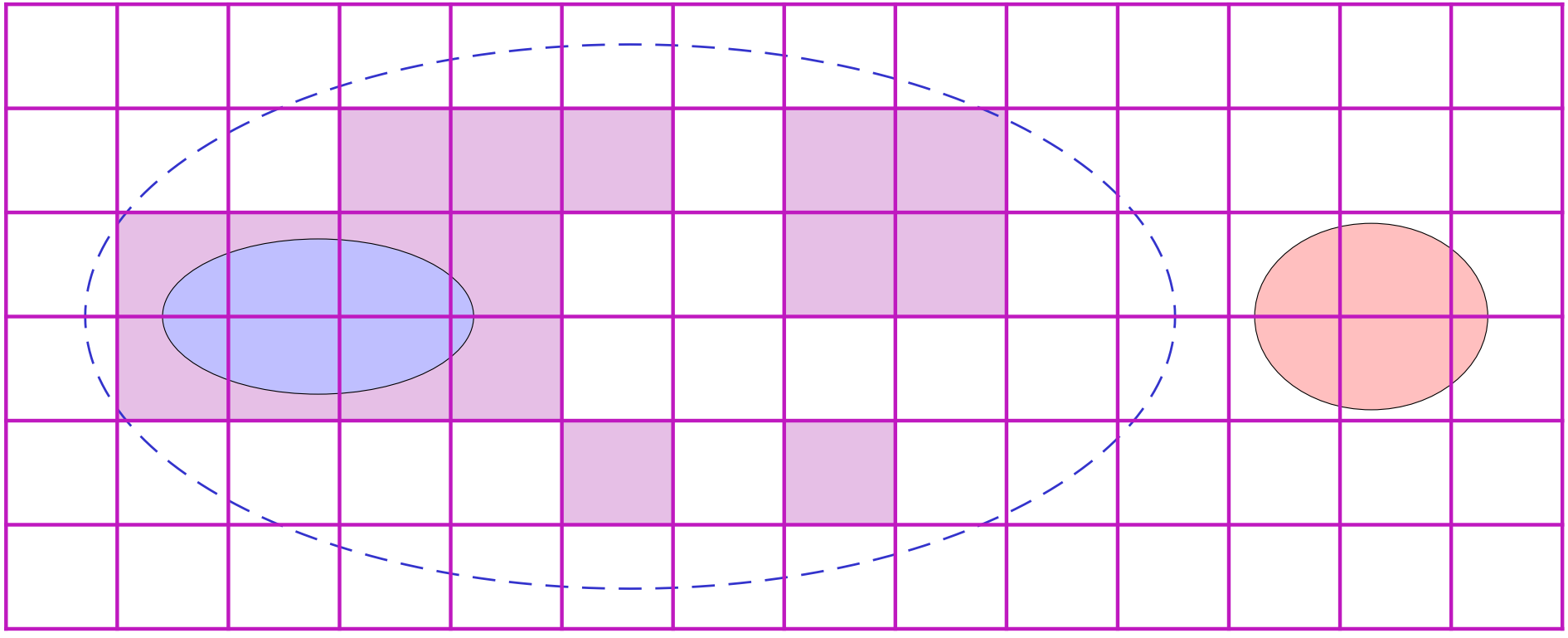
- Raffinement: ajout de nouveaux prédicats
- Grille plus fine

Raffinement de l'abstraction par prédicats



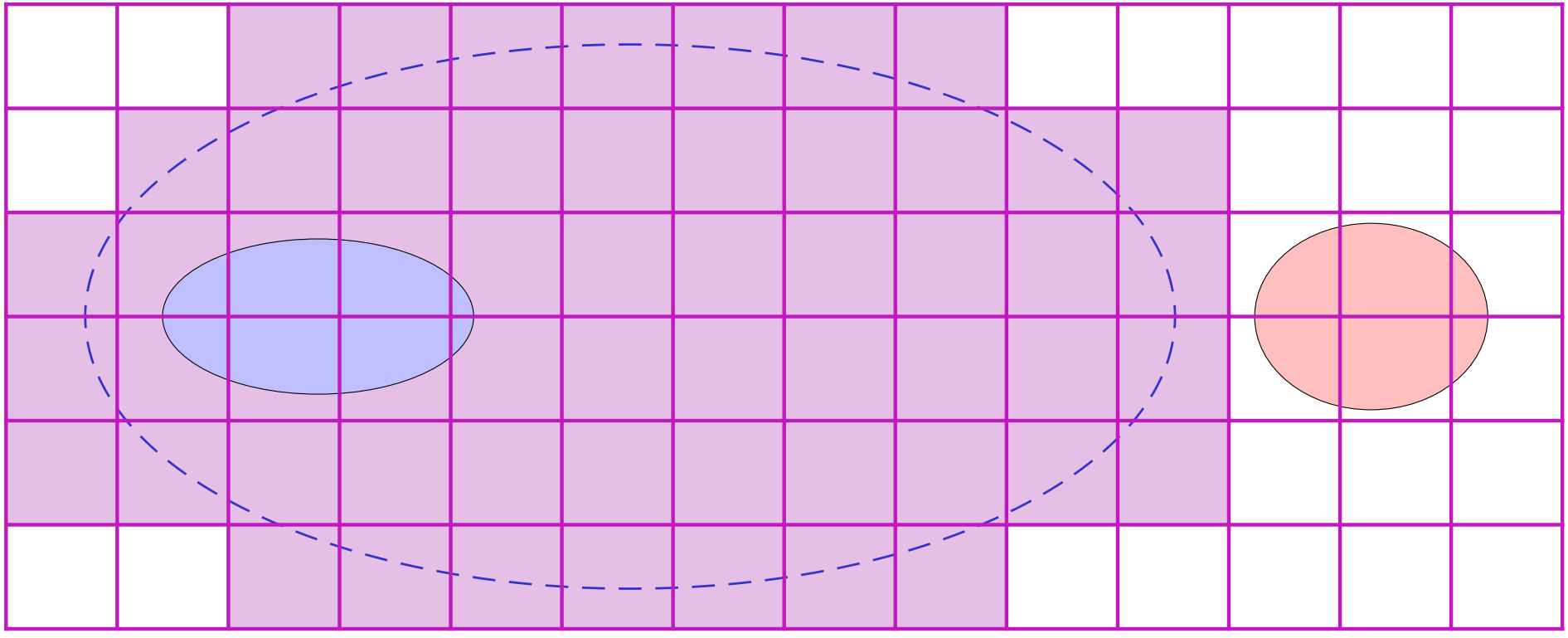
- Raffinement: ajout de nouveaux prédicats
- Grille plus fine

Raffinement de l'abstraction par prédicats



- Raffinement: ajout de nouveaux prédicats
- Grille plus fine

Raffinement de l'abstraction par prédicats



- Raffinement: ajout de nouveaux prédicats
- Grille plus fine
- L'abstraction raffinée est sûre!

Abstraction-Vérification-Raffinement

- Trois phases intégrée dans une boucle complètement automatique
  - Aucun faux négatif

Abstraction-Vérification-Raffinement

- Trois phases intégrée dans une boucle complètement automatique
  - Aucun faux négatif
- Vérification automatique des contre-exemples abstraits
  - pour les propriétés de sûreté, revient à de la simulation symbolique

Abstraction-Vérification-Raffinement

- Trois phases intégrée dans une boucle complètement automatique
  - Aucun faux négatif
- Vérification automatique des contre-exemples abstraits
  - pour les propriétés de sûreté, revient à de la simulation symbolique
- Raffinement automatique
  - Nouveaux prédicats inférés automatiquement des contre-exemples erronés

Abstraction-Vérification-Raffinement (2)

Input: Système (progr. ou modèle) S et prédicats initiaux $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$

Abstraction-Vérification-Raffinement (2)

Input: Système (progr. ou modèle) S et prédicats initiaux $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$

1. calculer l'abstraction (finie) par prédicats A de S vis-à-vis des prédicats $\mathcal{P}$

Abstraction-Vérification-Raffinement (2)

Input: Système (progr. ou modèle) S et prédicats initiaux $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$

1. calculer l'abstraction (finie) par prédicats A de S vis-à-vis des prédicats $\mathcal{P}$
2. vérifier A
   - **si** A is correcte **alors retourner** S est correct
   - **sinon** extraire un chemin (abstrait) d'erreur π

Abstraction-Vérification-Raffinement (2)

Input: Système (progr. ou modèle) S et prédicats initiaux $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$

1. calculer l'abstraction (finie) par prédicats A de S vis-à-vis des prédicats $\mathcal{P}$
2. vérifier A
   - **si** A is correcte **alors retourner** S est correct
   - **sinon** extraire un chemin (abstrait) d'erreur π
3. déterminer si π est un chemin réalisable dans S
   - **si** ??? **alors retourner** ???
   - **si** π est réalisable dans S **alors retourner** S est incorrect

Abstraction-Vérification-Raffinement (2)

Input: Système (progr. ou modèle) S et prédicats initiaux $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$

1. calculer l'abstraction (finie) par prédicats A de S vis-à-vis des prédicats $\mathcal{P}$
2. vérifier A
   - **si** A is correcte **alors retourner** S est correct
   - **sinon** extraire un chemin (abstrait) d'erreur π
3. déterminer si π est un chemin réalisable dans S
   - **si** ??? **alors retourner** ???
   - **si** π est réalisable dans S **alors retourner** S est incorrect
4. raffinement
   - extraire une "explication" de non-réalisabilité de π sous la forme de nouveaux prédicats $P'_1, P'_2, \ldots, P'_k$
   - $\mathcal{P} := \mathcal{P} \cup \{P'_1, P'_2, \ldots, P'_k\}$
5. **aller à** 1.

Outline

1. Introduction
2. Abstraction
3. Interprétation abstraite
4. Abstraction par prédicats
5. **Raffinement guidé par contre-exemple**
6. Perspectives

Validation des contre-exemples

- l'abstraction a le même graphe de contrôle que le système concret
  - abstraction de données
- Simulation symbolique exacte le long du chemin d'erreur abstrait
- Pour des commandes gardées, on procède classiquement en arrière :
  - substitutions pour les affections
  - conjonctions avec les gardes
  - la réalisabilité du chemin dans le système concret revient à la satisfaisabilité du prédicat obtenu

Découverte des prédicats

- le chemin d'erreur abstrait n'est pas réalisable dans le système concret
- trouver des prédicats permettant de séparer les bons états des mauvais états
- Pour des commandes gardées, on peut procéder ainsi :
- Garder les substitutions explicites
  - introduire de nouvelles variables (quantifiées existentiellement)
  - toutes les opérations apparaiisent explicitement
- Demander une peuve de non-satisfaisabilité
- Prendre les prédicats apparaissant dans la preuve

Outline

1. Introduction
2. Abstraction
3. Interprétation abstraite
4. Abstraction par prédicats
5. Raffinement guidé par contre-exemple
6. Perspectives

Conclusion

- Automatisation complète de la boucle abstraction-vérification-raffinement
- Abstraction calculée à l'aide d'outils existants (procédures de décision)
- Applications :
  - circuits
  - code (pilotes de noyau)
  - systèmes hybrides
- permet de produire des certificats (PCC)
- Recherche très active

Perspectives

- Découverte des prédicats
  - Leur nombre doit rester petit
  - Techniques d'accélération afin d'éviter de dérouler les boucles
- Algorithmique de l'abstraction par prédicats
  - BDD interprétés
  - limiter le nombre d'appels au prouveur
- Applications à d'autres modèles (FIFO, hétérogènes)
- Combinaison avec les techniques de vérification symbolique
- Complétude de l'approche

References

[BMMR01] T. Ball, R. Majumdar, T. D. Millstein, and S. K. Rajamani. Automatic predicate abstraction of c programs. In *Proc. ACM SIGPLAN '01 Conf. Programming Language Design and Implementation (PLDI'01), Snowbird, UT, USA, June 2001*, pages 203–213. ACM Press, 2001.

[BPR01] T. Ball, A. Podelski, and S. K. Rajamani. Boolean and cartesian abstraction for model checking c programs. In *Proc. 7th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2001), Genova, Italy, Apr. 2001*, volume 2031 of *Lecture Notes in Computer Science*, pages 268–283. Springer, 2001.

[BR01] T. Ball and S. K. Rajamani. Automatically validating temporal safety properties of interfaces. In *Proc. 8th Int. SPIN Workshop (SPIN'2001), Toronto, Canada, May 2001*, volume 2057 of *Lecture Notes in Computer Science*, pages 103–122. Springer, 2001.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM Symp. Principles of Programming Languages, San Antonio, TX, USA*, pages 269–282. ACM Press, 1979.

[CGJ+00] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[DD01] S. Das and D. L. Dill. Successive approximation of abstract transition relations. In *Proc. 16th IEEE Symp. Logic in Computer Science (LICS'2001), Boston, MA, USA, June 2001*, pages 51–60, 2001.

[DDP99] S. Das, D. L. Dill, and S. Park. Experience with predicate abstraction. In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99), Trento, Italy, July 1999*, volume 1633 of *Lecture Notes in Computer Science*, pages 160–171. Springer, 1999.

[DGG97] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1997.

[GS97] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Proc. 9th Int. Conf. Computer Aided Verification (CAV'97), Haifa, Israel, June 1997*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1997.

[HJMS02] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *Proc. 29th ACM Symp. Principles of Programming Languages (POPL'2002), Portland, OR, USA, Jan. 2002*, pages 58–70. ACM Press, 2002.

[KPV01] Y. Kesten, A. Pnueli, and M. Y. Vardi. Verification by augmented abstraction: The automata-theoretic view. *Journal of Computer and System Sciences*, 62(4):668–690, 2001.

[LBBO01] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre. Incremental verification by abstraction. In *Proc. 7th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2001), Genova, Italy, Apr. 2001*, volume 2031 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 2001.

[LGS+95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):1–44, 1995.