Analyse syntaxique appliquée

Adrien Guatto

Compilation M1

Introduction

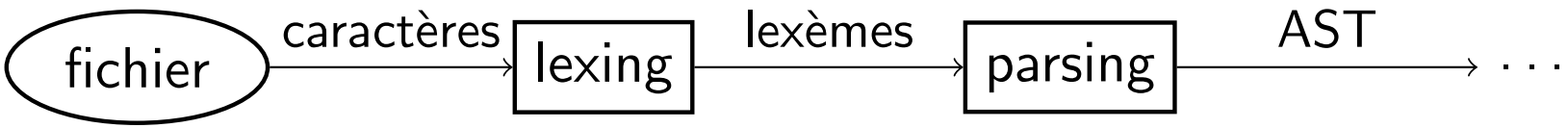
L'analyse lexicale et syntaxique, pourquoi ?

- Un compilateur manipule de la syntaxe abstraite.
- Un programmeur écrit du code source au format texte.

Introduction

L'analyse lexicale et syntaxique, pourquoi ?

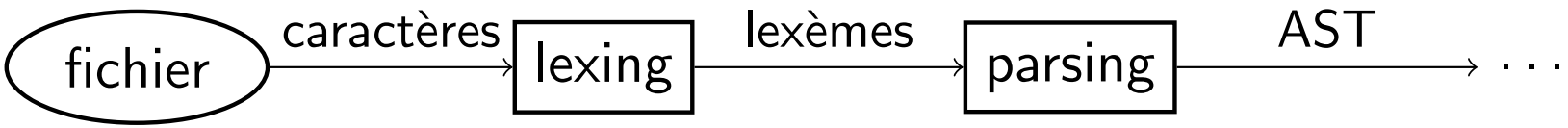
- Un compilateur manipule de la syntaxe abstraite.
- Un programmeur écrit du code source au format texte.



Introduction

L'analyse lexicale et syntaxique, pourquoi ?

- Un compilateur manipule de la syntaxe abstraite.
- Un programmeur écrit du code source au format texte.



Comment écrire un analyseur lexical ? Un analyseur syntaxique ?

- À la main, comme on l'a vu dans `marthe.ml`.
  - Analyse syntaxique par *parcours récursif descendant* dit "LL(1)".
- Via des générateurs de code : ocamllex et Menhir.
  - Des formalismes *déclaratifs* plutôt que des parcours explicites.
  - Analyse syntaxique par *parcours ascendant* dit "LR(1)".

Cette séance

Contexte

- Les générateurs de code transforment une description déclarative de la syntaxe du langage en un lexeur ou parseur.
- L'analyse syntaxique est significativement plus complexe que l'analyse lexicale, et exige un certain doigté.
- Les générateurs d'analyseurs syntaxiques ne sont pas capables de traiter une grammaire arbitraire (on verra que cela n'a pas de sens !).

Cette séance

Contexte

- Les générateurs de code transforment une description déclarative de la syntaxe du langage en un lexeur ou parseur.
- L'analyse syntaxique est significativement plus complexe que l'analyse lexicale, et exige un certain doigté.
- Les générateurs d'analyseurs syntaxiques ne sont pas capables de traiter une grammaire arbitraire (on verra que cela n'a pas de sens !).

L'objectif de la séance d'aujourd'hui

Comprendre suffisamment de théorie des grammaires pour être capable d'utiliser Menhir de façon productive.

Cette séance

Contexte

- Les générateurs de code transforment une description déclarative de la syntaxe du langage en un lexeur ou parseur.
- L'analyse syntaxique est significativement plus complexe que l'analyse lexicale, et exige un certain doigté.
- Les générateurs d'analyseurs syntaxiques ne sont pas capables de traiter une grammaire arbitraire (on verra que cela n'a pas de sens !).

L'objectif de la séance d'aujourd'hui

Comprendre suffisamment de théorie des grammaires pour être capable d'utiliser Menhir de façon productive.

Références bibliographiques

- Modern Compiler Implementation d'Appel, chapitres 2 et surtout 3.
- Practical Parsing Techniques de Grune et Jacobs.

Outline

Rappels sur l'analyse lexicale

- On veut transformer une chaîne de caractères en liste de lexèmes.
- On peut écrire manuellement un analyseur lexical, cf. `marthe.ml`.
- C'est assez laborieux, surtout quand on doit transformer plusieurs caractères en un unique lexème.

Rappels sur l'analyse lexicale

- On veut transformer une chaîne de caractères en liste de lexèmes.
- On peut écrire manuellement un analyseur lexical, cf. `marthe.ml`.
- C'est assez laborieux, surtout quand on doit transformer plusieurs caractères en un unique lexème.

Un exemple

"La syntaxe lexicale d'un nombre à virgule flottante est composée d'un signe optionnel, d'une mantisse, et d'un exposant optionnel. Le signe peut être '+' ou '-'. La mantisse est formée d'une suite de chiffres suivie d'un point suivi d'une suite de chiffres optionnelle. L'exposant peut être formé par le caractère 'e' ou 'E' suivi d'une suite finie de chiffres."

Rappels sur l'analyse lexicale

- On veut transformer une chaîne de caractères en liste de lexèmes.
- On peut écrire manuellement un analyseur lexical, cf. `marthe.ml`.
- C'est assez laborieux, surtout quand on doit transformer plusieurs caractères en un unique lexème.

Un exemple

"La syntaxe lexicale d'un nombre à virgule flottante est composée d'un signe optionnel, d'une mantisse, et d'un exposant optionnel. Le signe peut être '+' ou '-'. La mantisse est formée d'une suite de chiffres suivie d'un point suivi d'une suite de chiffres optionnelle. L'exposant peut être formé par le caractère 'e' ou 'E' suivi d'une suite finie de chiffres."

Vous connaissez le formalisme déclaratif adapté : les *expressions régulières*.

Analyse lexicale avec `ocamllex`

Voir le manuel d'OCaml, "Lexer and parser generators", §13.1 et §13.2.

```
{ header }
let ident = regexp ...
rule entrypoint [arg1... argn] =
  parse regexp { action }
      | ...
      | regexp { action }
and entrypoint [arg1... argn] =
  parse ...
and ...
{ trailer }
```

Analyse lexicale avec `ocamllex`

Voir le manuel d'OCaml, "Lexer and parser generators", §13.1 et §13.2.

```
{ header }
let ident = regexp ...
rule entrypoint [arg1... argn] =
  parse regexp { action }
      | ...
      | regexp { action }
and entrypoint [arg1... argn] =
  parse ...
and ...
{ trailer }
```

- Les actions d'une règle sont des expressions OCaml de même type.
  - Dans un compilateur, chaque action va produire un lexème.
- Les regexps sont testées dans l'ordre, comme le `match` d'OCaml.
- Le fichier `.mll` est traduit en un automate fini déterministe.
  - Chaque règle donne lieu à une fonction OCaml aux mêmes arguments.

De l'analyse lexicale à l'analyse syntaxique

- L'écriture d'un analyseur lexical est à peu près une formalité.
  - Les expressions régulières décrivent la forme des lexèmes avec concision.
  - Les actions associées à chaque lexème permettent d'exécuter du code arbitraire lors d'une transition de l'automate.

De l'analyse lexicale à l'analyse syntaxique

- L'écriture d'un analyseur lexical est à peu près une formalité.
  - Les expressions régulières décrivent la forme des lexèmes avec concision.
  - Les actions associées à chaque lexème permettent d'exécuter du code arbitraire lors d'une transition de l'automate.
- Les expressions régulières sont insuffisantes pour l'analyse syntaxique.
  - Par exemple : le langage $\{(^n)^n \mid n \in \mathbb{N}\}$ n'est pas régulier.
  - En plus concret : ne tentez pas de parser HTML avec des regexps !

De l'analyse lexicale à l'analyse syntaxique

- L'écriture d'un analyseur lexical est à peu près une formalité.
  - Les expressions régulières décrivent la forme des lexèmes avec concision.
  - Les actions associées à chaque lexème permettent d'exécuter du code arbitraire lors d'une transition de l'automate.
- Les expressions régulières sont insuffisantes pour l'analyse syntaxique.
  - Par exemple : le langage $\{(^n)^n \mid n \in \mathbb{N}\}$ n'est pas régulier.
  - En plus concret : ne tentez pas de parser HTML avec des regexps !
- Les *grammaires hors-contexte* sont strictement plus expressives.

Outline

Rappels sur les grammaires hors-contexte

Une *grammaire hors-contexte* $\mathcal{G} = (N_\mathcal{G}, T_\mathcal{G}, R_\mathcal{G}, S_\mathcal{G})$ est un quadruplet où

- $N_\mathcal{G}$ est un ensemble fini de *non-terminaux*,
- $T_\mathcal{G}$ est un ensemble fini de *terminaux* disjoint de $N_\mathcal{G}$,
- $R_\mathcal{G} : N_\mathcal{G} \to \mathcal{P}_{fin}((N_\mathcal{G} \cup T_\mathcal{G})^*)$ associe à $n \in N_\mathcal{G}$ ses *règles* $R_\mathcal{G}(n)$,
- $S_\mathcal{G} \in N_\mathcal{G}$ est un non-terminal distingué, le *symbole de départ*.

Rappels sur les grammaires hors-contexte

Une *grammaire hors-contexte* $\mathcal{G} = (N_\mathcal{G}, T_\mathcal{G}, R_\mathcal{G}, S_\mathcal{G})$ est un quadruplet où

- $N_\mathcal{G}$ est un ensemble fini de *non-terminaux*,
- $T_\mathcal{G}$ est un ensemble fini de *terminaux* disjoint de $N_\mathcal{G}$,
- $R_\mathcal{G} : N_\mathcal{G} \to \mathcal{P}_{fin}((N_\mathcal{G} \cup T_\mathcal{G})^*)$ associe à $n \in N_\mathcal{G}$ ses *règles* $R_\mathcal{G}(n)$,
- $S_\mathcal{G} \in N_\mathcal{G}$ est un non-terminal distingué, le *symbole de départ*.

L'*acceptation* $\models_\mathcal{G} \subseteq T_\mathcal{G}^* \times (T_\mathcal{G} \cup N_\mathcal{G})$ est la plus petite relation comprenant

$$\frac{}{t \models_\mathcal{G} t} \qquad \frac{w_1 \models_\mathcal{G} x_1 \quad \dots \quad w_n \models_\mathcal{G} x_n}{w_1 \cdot \ldots \cdot w_n \models_\mathcal{G} n} (x_1 \cdot \ldots \cdot x_n \in R_\mathcal{G}(n))$$

Rappels sur les grammaires hors-contexte

Une *grammaire hors-contexte* $\mathcal{G} = (N_\mathcal{G}, T_\mathcal{G}, R_\mathcal{G}, S_\mathcal{G})$ est un quadruplet où

- $N_\mathcal{G}$ est un ensemble fini de *non-terminaux*,
- $T_\mathcal{G}$ est un ensemble fini de *terminaux* disjoint de $N_\mathcal{G}$,
- $R_\mathcal{G} : N_\mathcal{G} \to \mathcal{P}_{fin}((N_\mathcal{G} \cup T_\mathcal{G})^*)$ associe à $n \in N_\mathcal{G}$ ses *règles* $R_\mathcal{G}(n)$,
- $S_\mathcal{G} \in N_\mathcal{G}$ est un non-terminal distingué, le *symbole de départ*.

L'*acceptation* $\models_\mathcal{G} \subseteq T_\mathcal{G}^* \times (T_\mathcal{G} \cup N_\mathcal{G})$ est la plus petite relation comprenant

$$\frac{}{t \models_\mathcal{G} t} \qquad \frac{w_1 \models_\mathcal{G} x_1 \quad \dots \quad w_n \models_\mathcal{G} x_n}{w_1 \cdot \ldots \cdot w_n \models_\mathcal{G} n} (x_1 \cdot \ldots \cdot x_n \in R_\mathcal{G}(n))$$

Le *langage* de $\mathcal{G}$, dénoté $\mathbb{L}(\mathcal{G})$, est défini par

$$\mathbb{L}(\mathcal{G}) \triangleq \{w \in T_\mathcal{G}^* \mid w \models_\mathcal{G} S_\mathcal{G}\}.$$

Exemple de grammaire hors-contexte

```
start ::= exp EOF
exp ::= exp PLUS exp | exp STAR exp | INT
```

Exemple de grammaire hors-contexte

```
start ::= exp EOF
exp ::= exp PLUS exp | exp STAR exp | INT
```

On définit $\mathcal{G}_{exp} \triangleq (\{\texttt{start}, \texttt{exp}\}, \{\texttt{INT}, \texttt{PLUS}, \texttt{STAR}, \texttt{EOF}\}, R, \texttt{start})$ avec

$$R(\texttt{start}) = \{\texttt{exp EOF}\},$$
$$R(\texttt{exp}) = \{\texttt{INT}, \texttt{exp PLUS exp}, \texttt{exp STAR exp}\}.$$

Exemple de grammaire hors-contexte

```
start ::= exp EOF
exp ::= exp PLUS exp | exp STAR exp | INT
```

On définit $\mathcal{G}_{exp} \triangleq (\{\texttt{start}, \texttt{exp}\}, \{\texttt{INT}, \texttt{PLUS}, \texttt{STAR}, \texttt{EOF}\}, R, \texttt{start})$ avec

$$R(\texttt{start}) = \{\texttt{exp EOF}\},$$
$$R(\texttt{exp}) = \{\texttt{INT}, \texttt{exp PLUS exp}, \texttt{exp STAR exp}\}.$$

On a $w \triangleq \texttt{INT PLUS INT STAR INT EOF} \in \mathbb{L}(\mathcal{G}_{exp})$ car $w \models_{\mathcal{G}_{exp}} \texttt{start}$.

$$\dfrac{\dfrac{\dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}} \quad \dfrac{}{\texttt{PLUS} \models_{\mathcal{G}_{exp}} \texttt{PLUS}} \quad \dfrac{\dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}} \quad \dfrac{}{\texttt{STAR} \models_{\mathcal{G}_{exp}} \texttt{STAR}} \quad \dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}}}{\texttt{INT STAR INT} \models_{\mathcal{G}_{exp}} \texttt{exp}}}{\texttt{INT PLUS INT STAR INT} \models_{\mathcal{G}_{exp}} \texttt{exp}} \quad \dfrac{}{\texttt{EOF} \models_{\mathcal{G}_{exp}} \texttt{EOF}}}{\texttt{INT PLUS INT STAR INT EOF} \models_{\mathcal{G}_{exp}} \texttt{start}}$$

Exemple de grammaire hors-contexte

```
start ::= exp EOF
exp ::= exp PLUS exp | exp STAR exp | INT
```

On définit $\mathcal{G}_{exp} \triangleq (\{\texttt{start}, \texttt{exp}\}, \{\texttt{INT}, \texttt{PLUS}, \texttt{STAR}, \texttt{EOF}\}, R, \texttt{start})$ avec

$$R(\texttt{start}) = \{\texttt{exp EOF}\},$$
$$R(\texttt{exp}) = \{\texttt{INT}, \texttt{exp PLUS exp}, \texttt{exp STAR exp}\}.$$

On a $w \triangleq \texttt{INT PLUS INT STAR INT EOF} \in \mathbb{L}(\mathcal{G}_{exp})$ car $w \models_{\mathcal{G}_{exp}} \texttt{start}$.

$$\dfrac{\dfrac{\dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}} \quad \dfrac{}{\texttt{PLUS} \models_{\mathcal{G}_{exp}} \texttt{PLUS}} \quad \dfrac{\dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}} \quad \dfrac{}{\texttt{STAR} \models_{\mathcal{G}_{exp}} \texttt{STAR}} \quad \dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}}}{\texttt{INT STAR INT} \models_{\mathcal{G}_{exp}} \texttt{exp}}}{\texttt{INT PLUS INT STAR INT} \models_{\mathcal{G}_{exp}} \texttt{exp}} \quad \dfrac{}{\texttt{EOF} \models_{\mathcal{G}_{exp}} \texttt{EOF}}}{\texttt{INT PLUS INT STAR INT EOF} \models_{\mathcal{G}_{exp}} \texttt{start}}$$

Une remarque clé

La *dérivation* ci-dessus peut être vue comme un arbre de syntaxe abstraite.

Grammaires ambigües

Un premier problème

- Une *dérivation* décrit un arbre de syntaxe abstraite.
- Donc, un analyseur syntaxique ne s'intéresse pas seulement à l'acceptation mais aussi aux dérivations qui y mènent.
- Certaines grammaires, comme $\mathcal{G}_{exp}$, sont *ambigües* : le même mot peut être accepté par des dérivations distinctes.

Grammaires ambigües

Un premier problème

- Une *dérivation* décrit un arbre de syntaxe abstraite.
- Donc, un analyseur syntaxique ne s'intéresse pas seulement à l'acceptation mais aussi aux dérivations qui y mènent.
- Certaines grammaires, comme $\mathcal{G}_{exp}$, sont *ambigües* : le même mot peut être accepté par des dérivations distinctes.

Le mot `INT PLUS INT STAR INT EOF` est accepté par une autre dérivation.

$$\dfrac{\dfrac{\dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}} \quad \dfrac{}{\texttt{PLUS} \models_{\mathcal{G}_{exp}} \texttt{PLUS}} \quad \dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}}}{\texttt{INT PLUS INT} \models_{\mathcal{G}_{exp}} \texttt{exp}} \quad \dfrac{\dfrac{}{\texttt{STAR} \models_{\mathcal{G}_{exp}} \texttt{STAR}} \quad \dfrac{}{\texttt{INT} \models_{\mathcal{G}_{exp}} \texttt{INT}}}{\texttt{INT PLUS INT STAR INT} \models_{\mathcal{G}_{exp}} \texttt{exp}} \quad \dfrac{}{\texttt{EOF} \models_{\mathcal{G}_{exp}} \texttt{EOF}}}{\texttt{INT PLUS INT STAR INT EOF} \models_{\mathcal{G}_{exp}} \texttt{start}}$$

Cette dérivation correspond à un arbre de syntaxe distincte du précédent.

Questions d'implémentation

Les problèmes de l'analyse syntaxique à base de grammaire

- Étant donnée $\mathcal{G}$, peut-on, pour tout $w \in T_{\mathcal{G}}^*$, déterminer si $w \models_{\mathcal{G}} S_{\mathcal{G}}$?
- Le faire efficacement ? (La performance d'un compilateur importe !)
- S'assurer statiquement que $\mathcal{G}$ n'est pas ambigüe ?

Questions d'implémentation

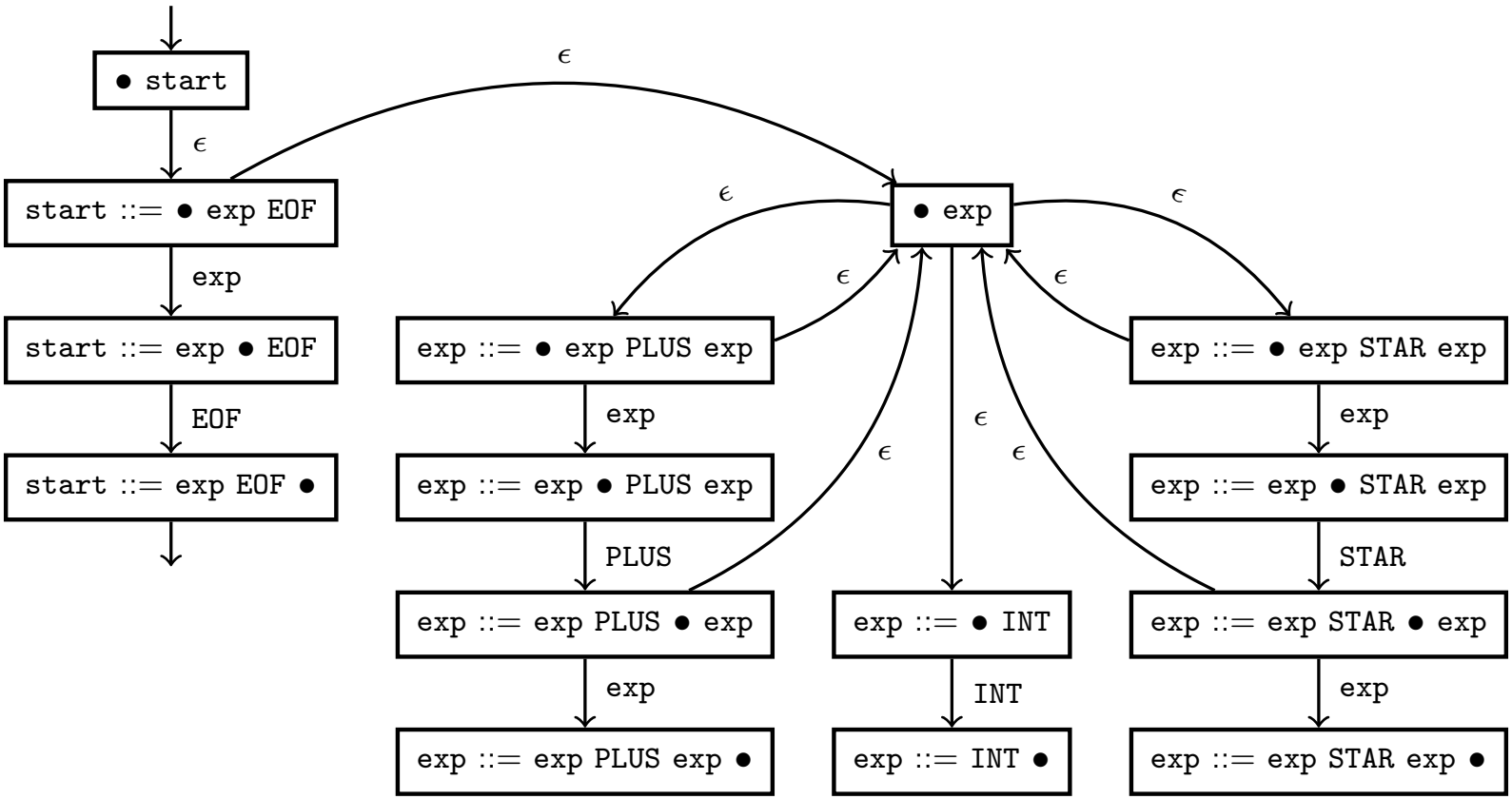
Les problèmes de l'analyse syntaxique à base de grammaire

- Étant donnée $\mathcal{G}$, peut-on, pour tout $w \in T_{\mathcal{G}}^*$, déterminer si $w \models_{\mathcal{G}} S_{\mathcal{G}}$?
- Le faire efficacement ? (La performance d'un compilateur importe !)
- S'assurer statiquement que $\mathcal{G}$ n'est pas ambigüe ?

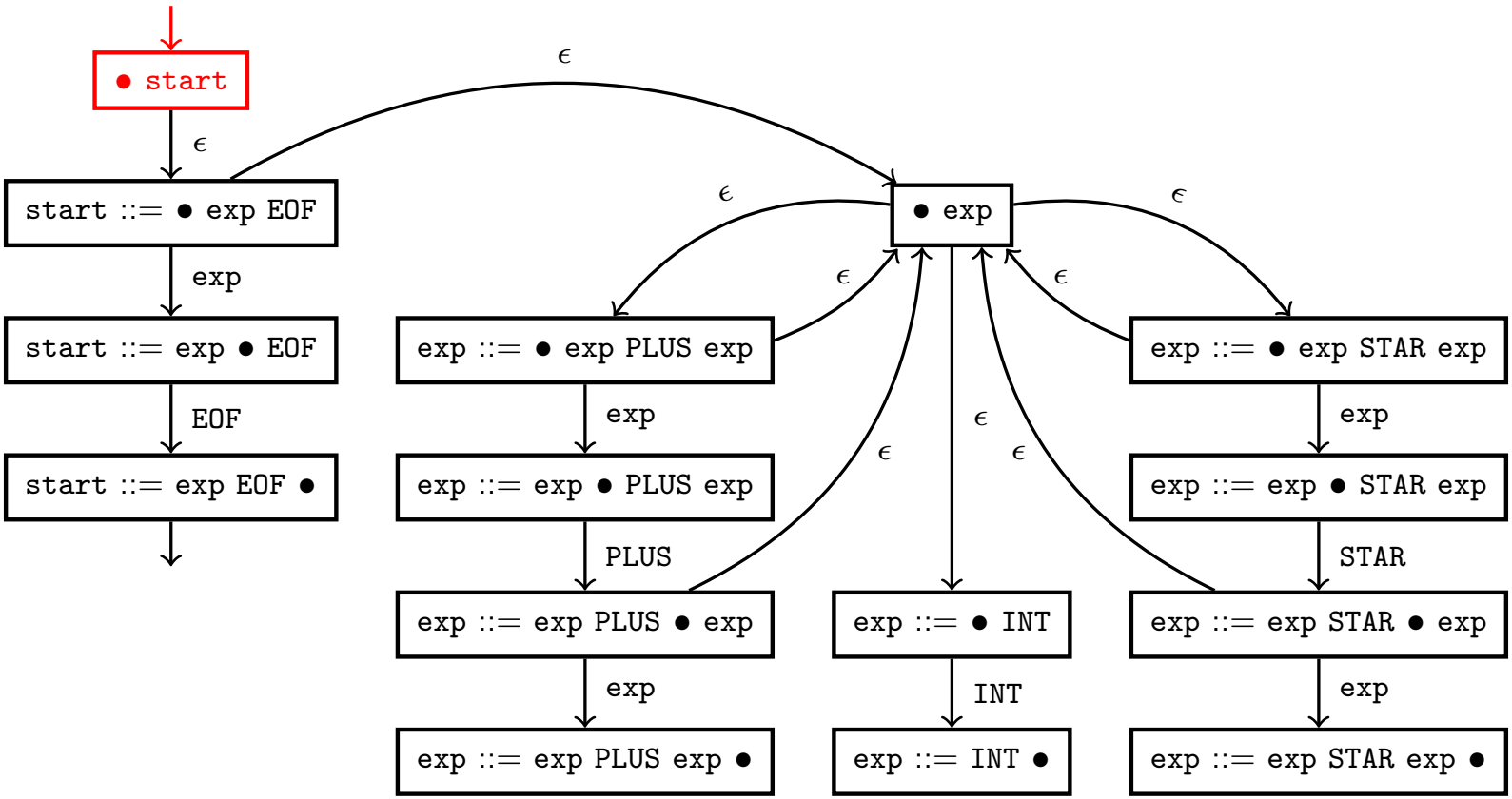
Vers un algorithme

- Une mauvaise idée : essayer toutes les dérivations possibles !
- Une bonne idée : utiliser un *automate à pile*.

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$

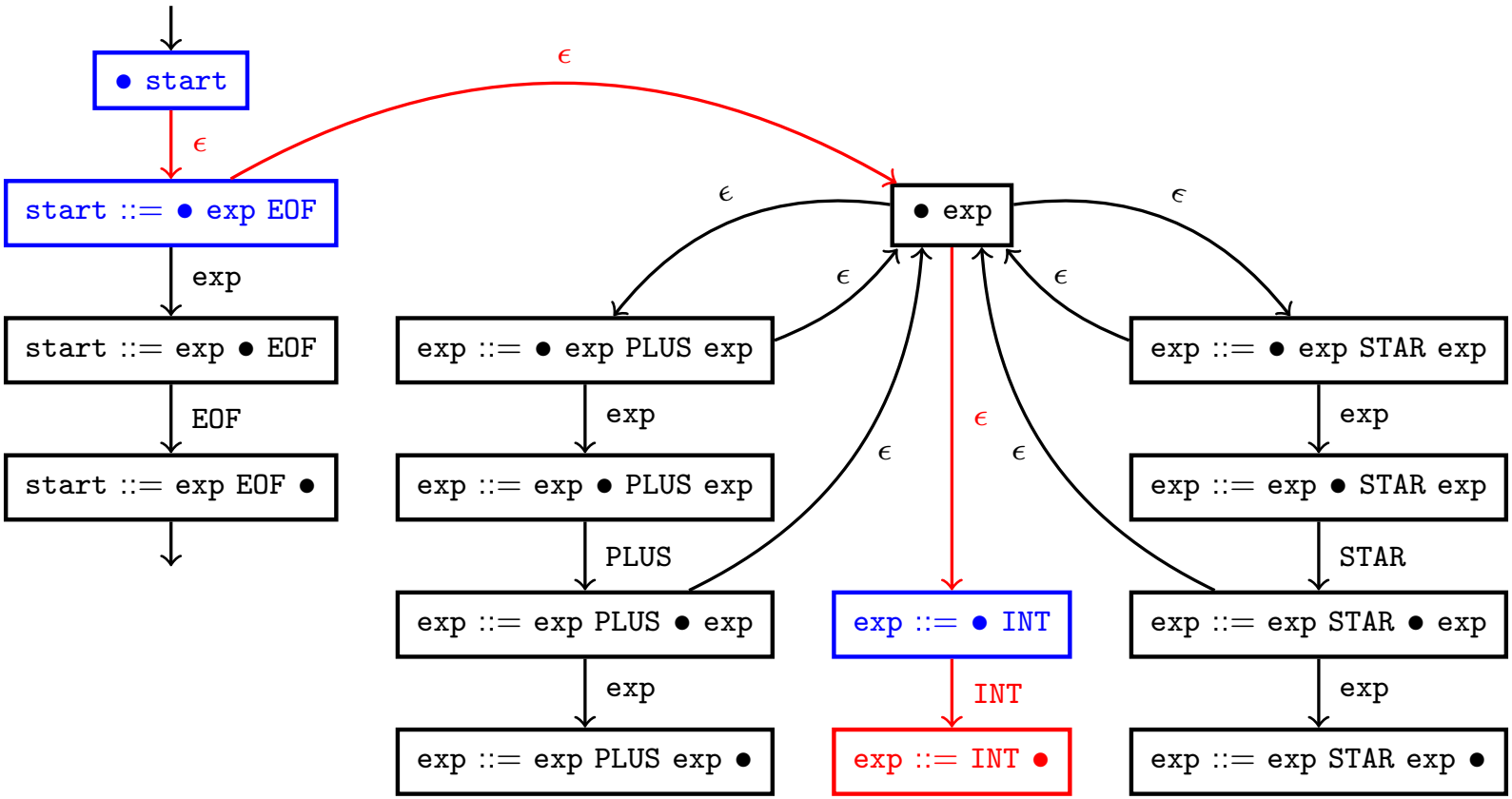


Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



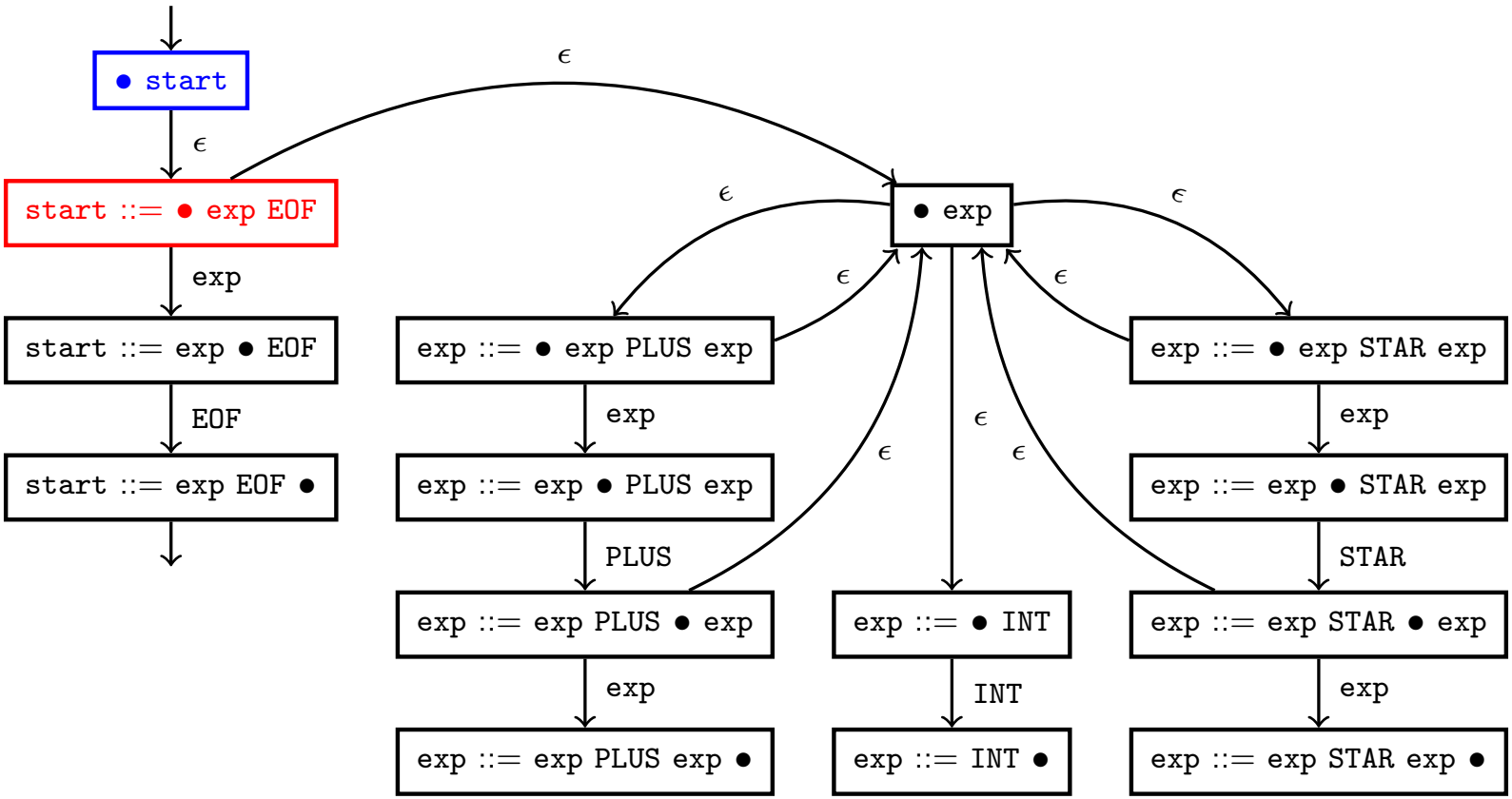
INT PLUS INT STAR INT EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



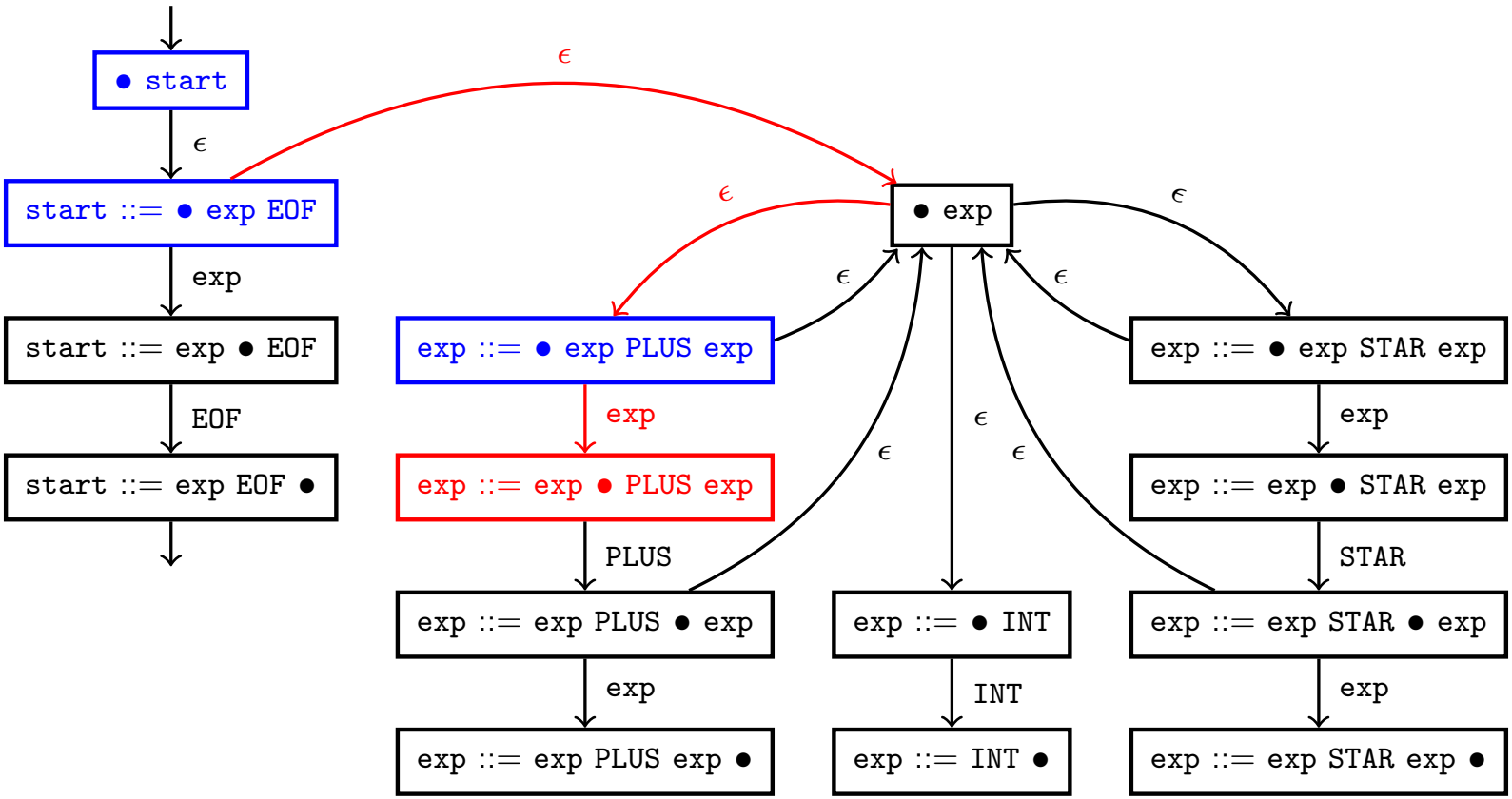
Décaler PLUS INT STAR INT EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



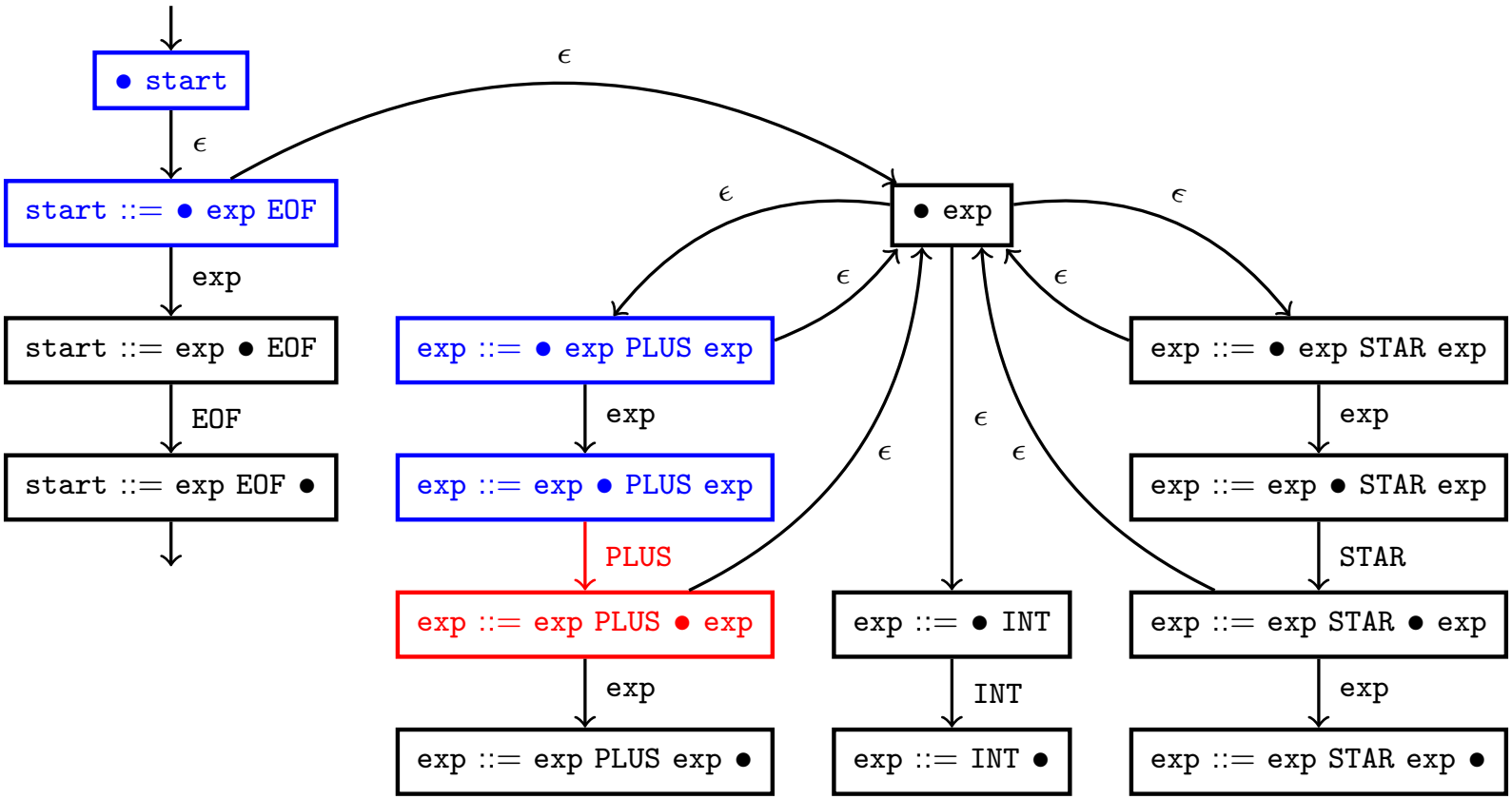
Réduire exp PLUS INT STAR INT EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



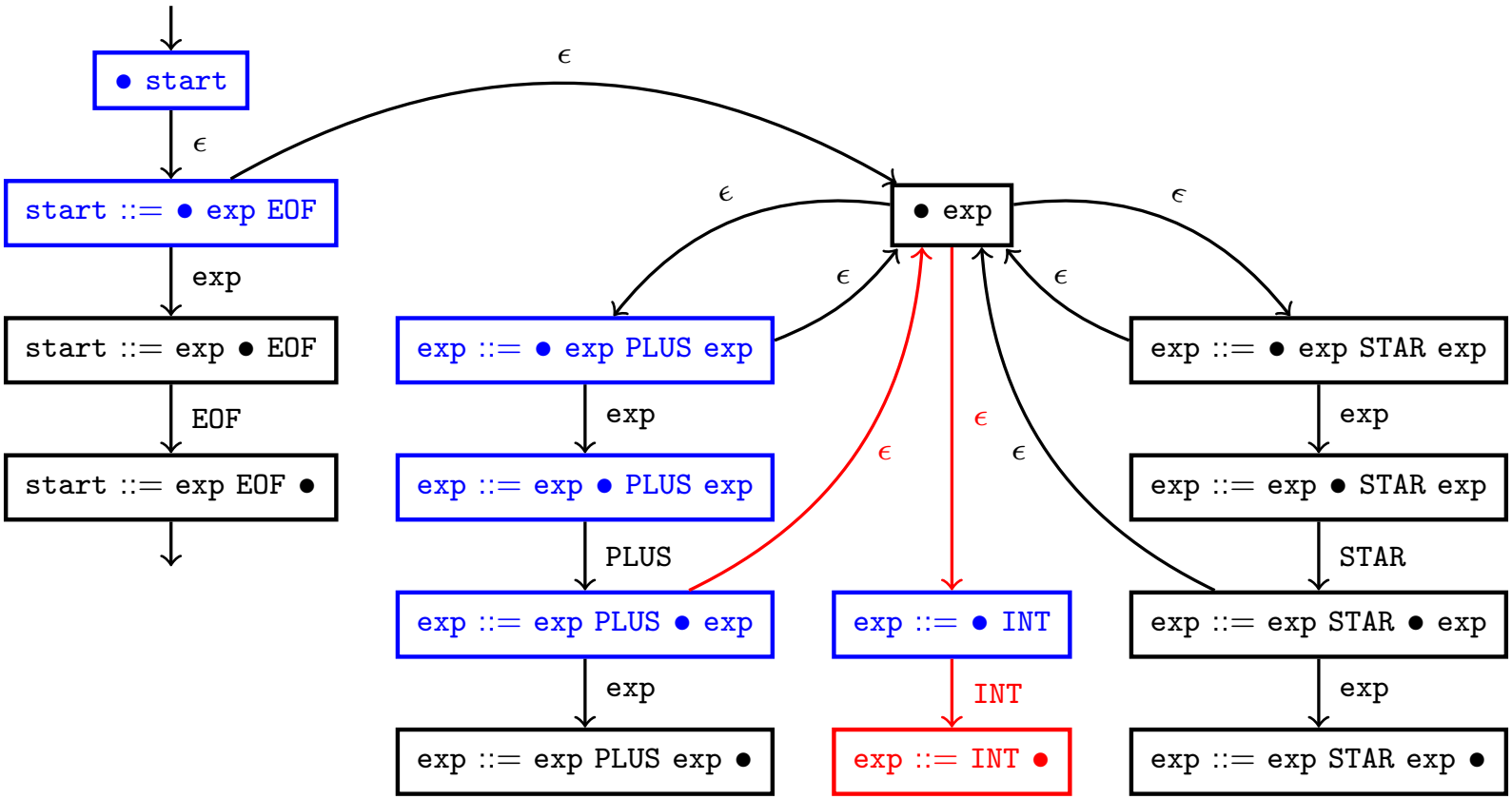
Décaler PLUS INT STAR INT EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



Décaler INT STAR INT EOF

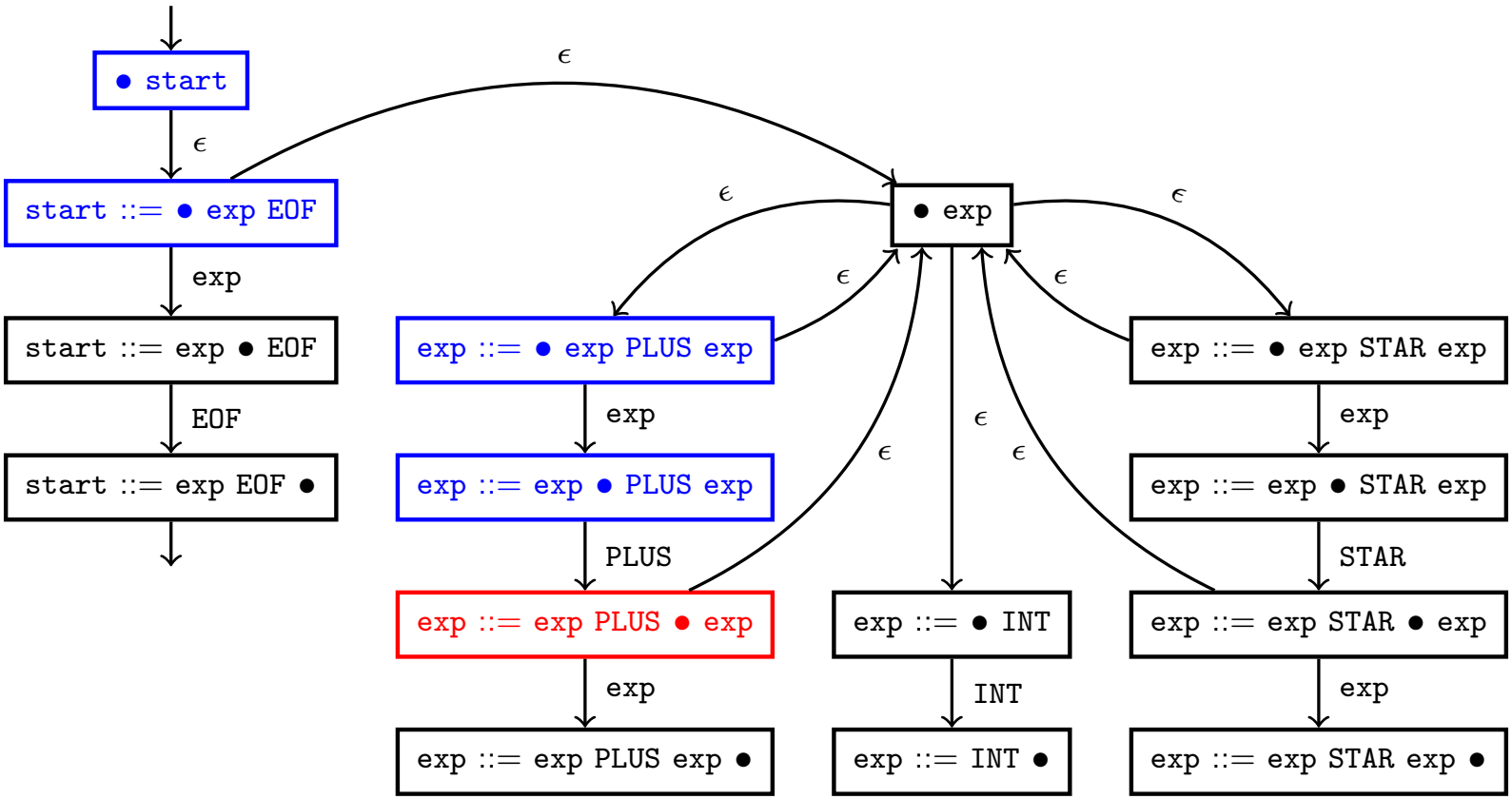
Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



Décaler

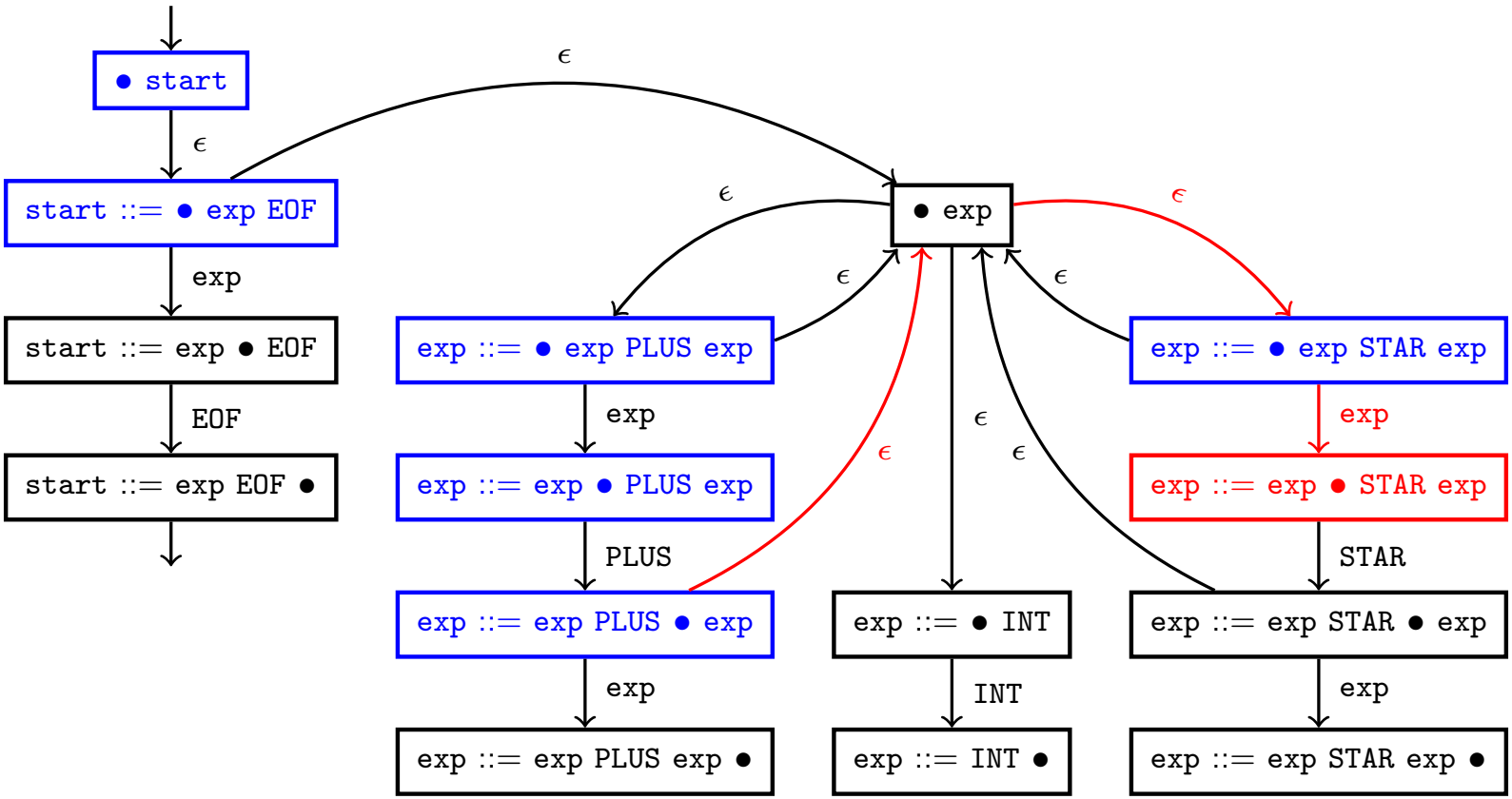
STAR INT EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



Réduire exp STAR INT EOF

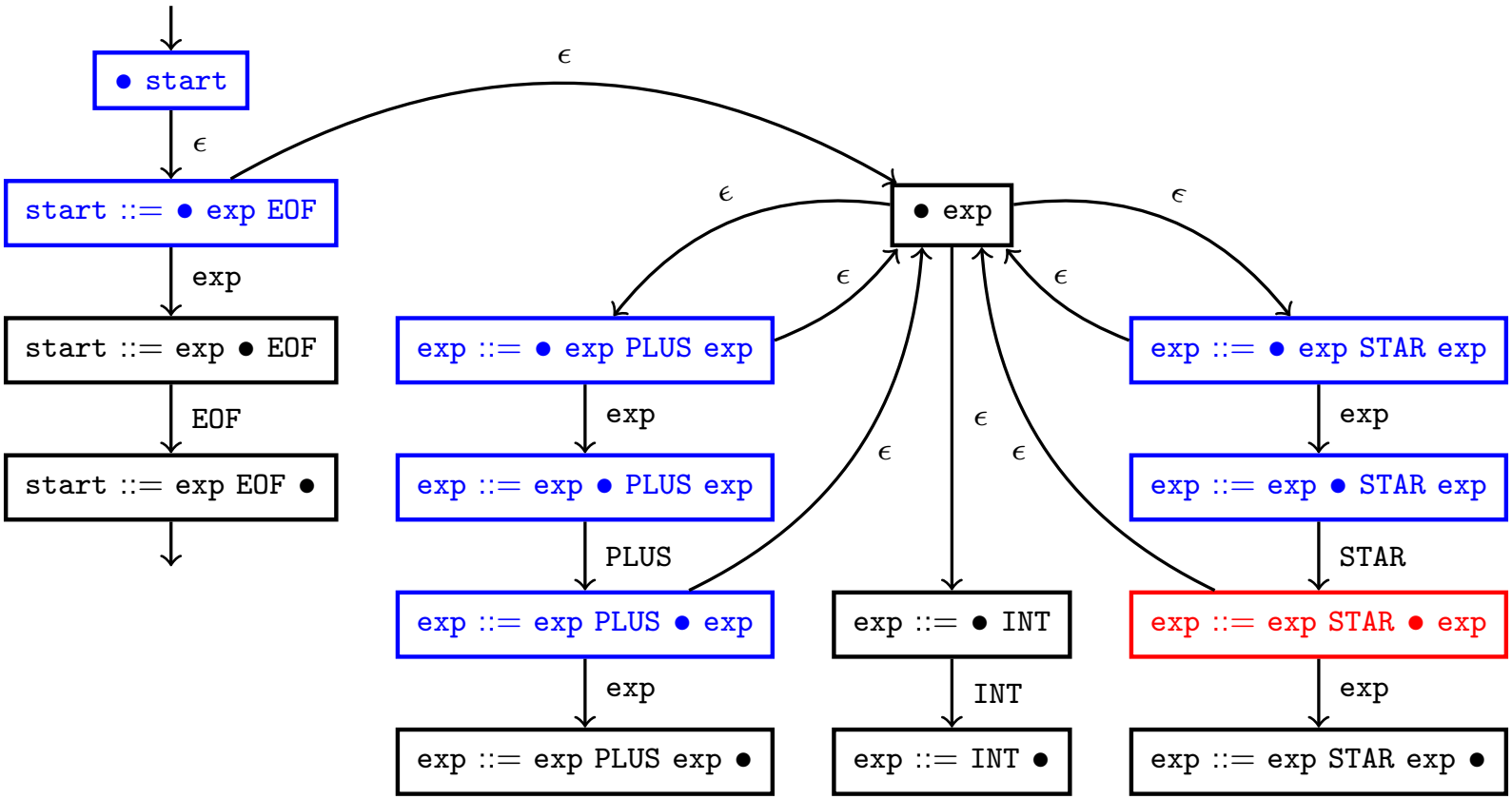
Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



Décaler

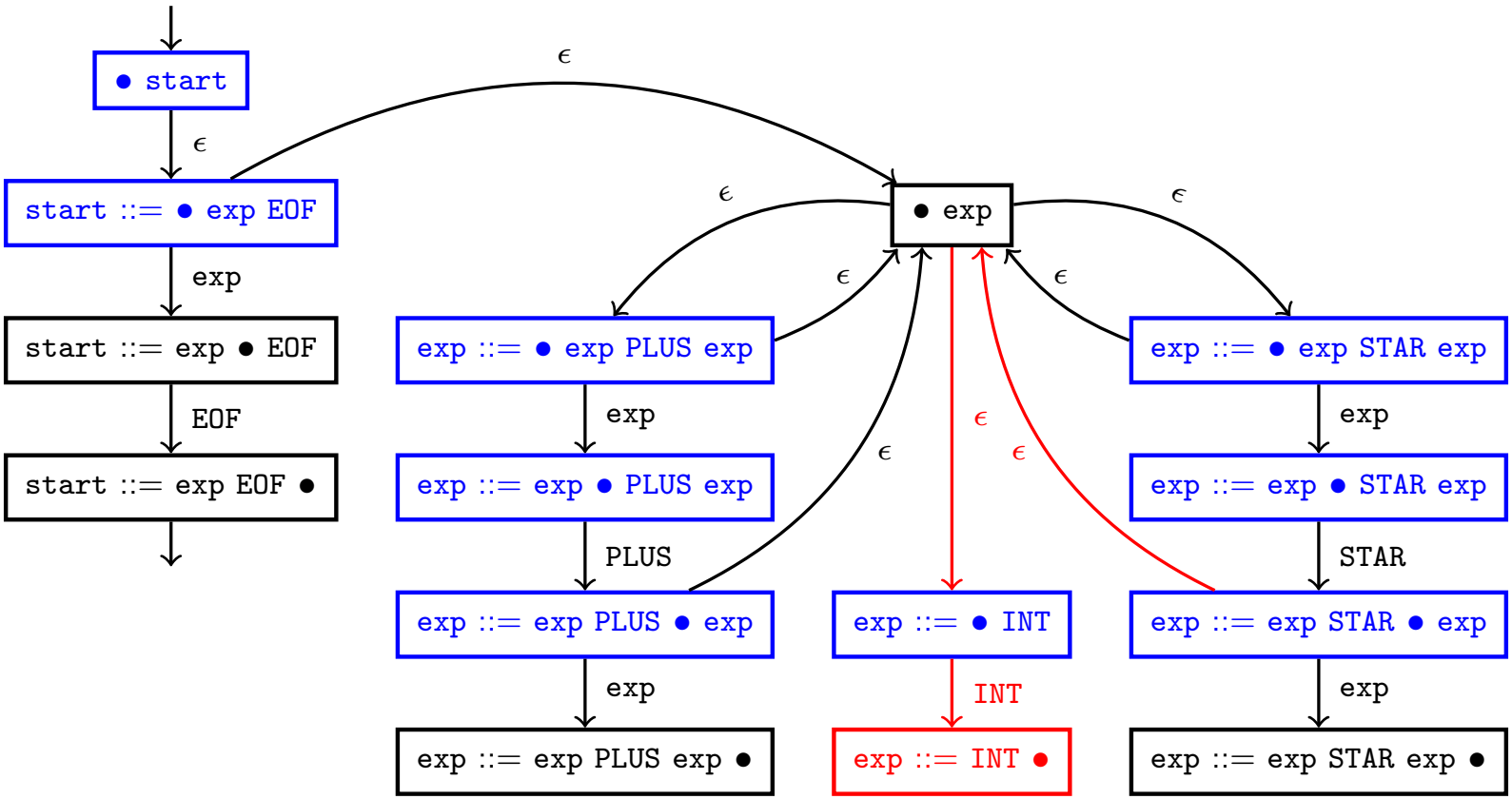
STAR INT EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



Décaler INT EOF

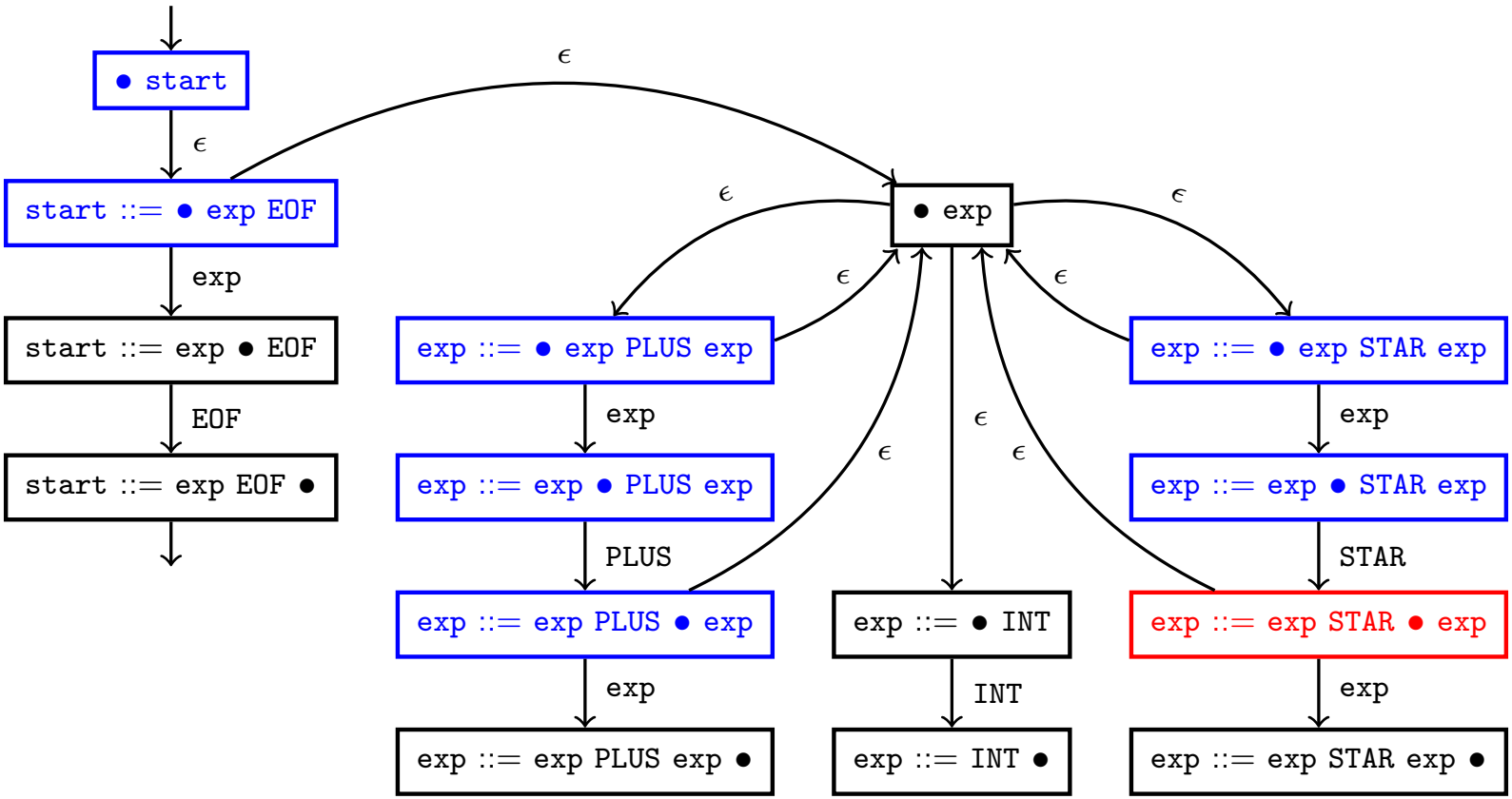
Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



Décaler

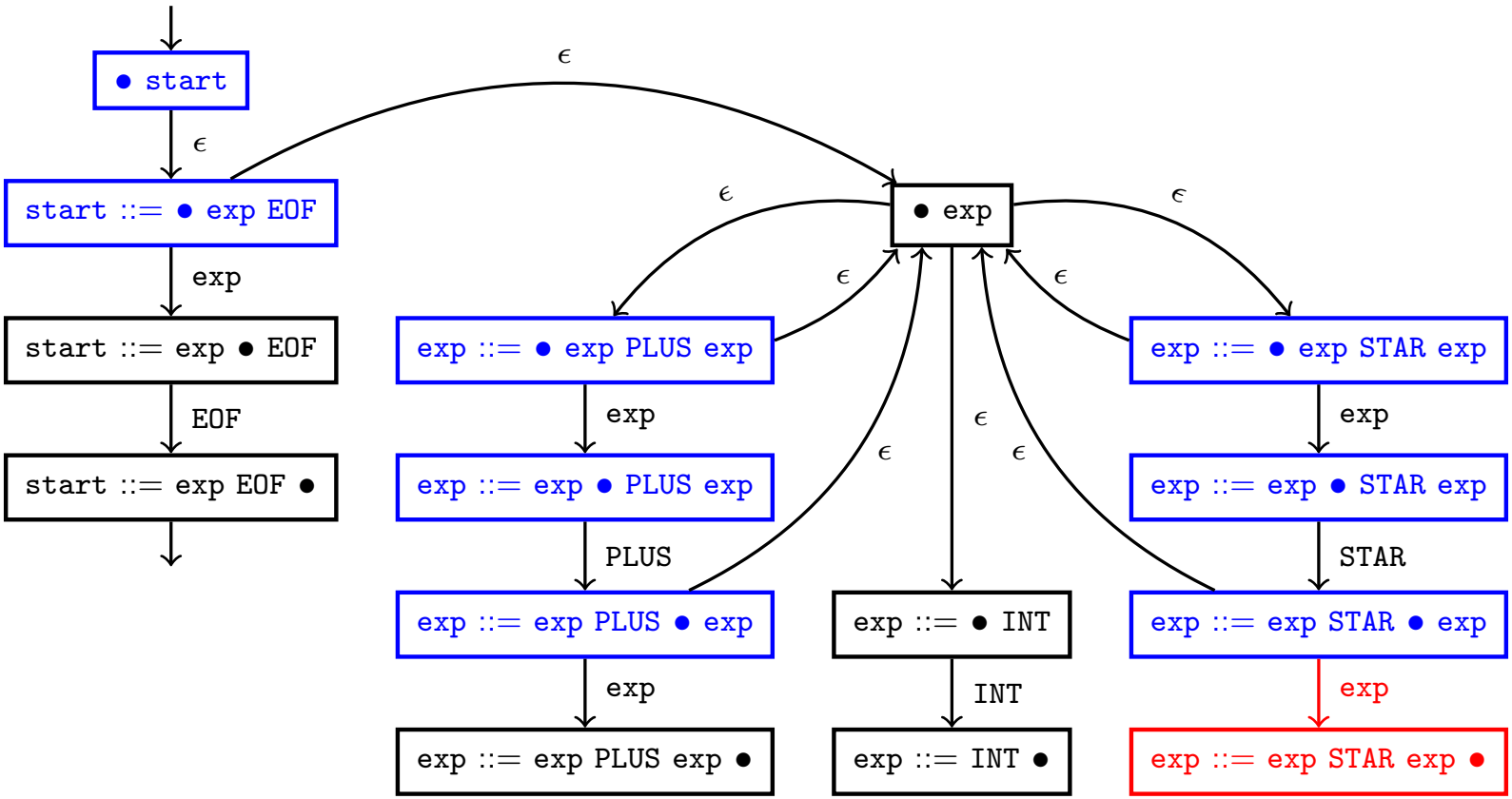
EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



Réduire exp EOF

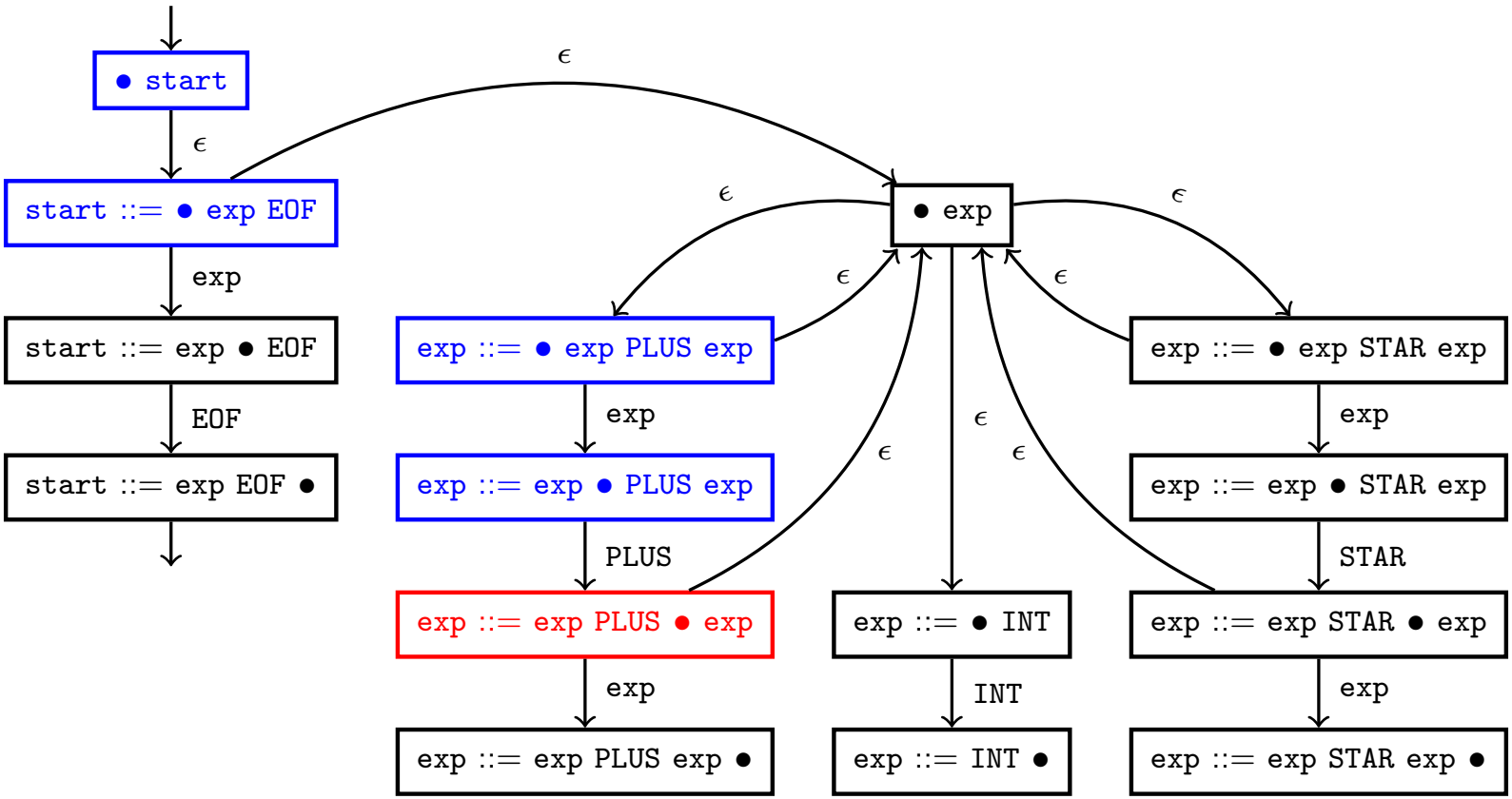
Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



Décaler

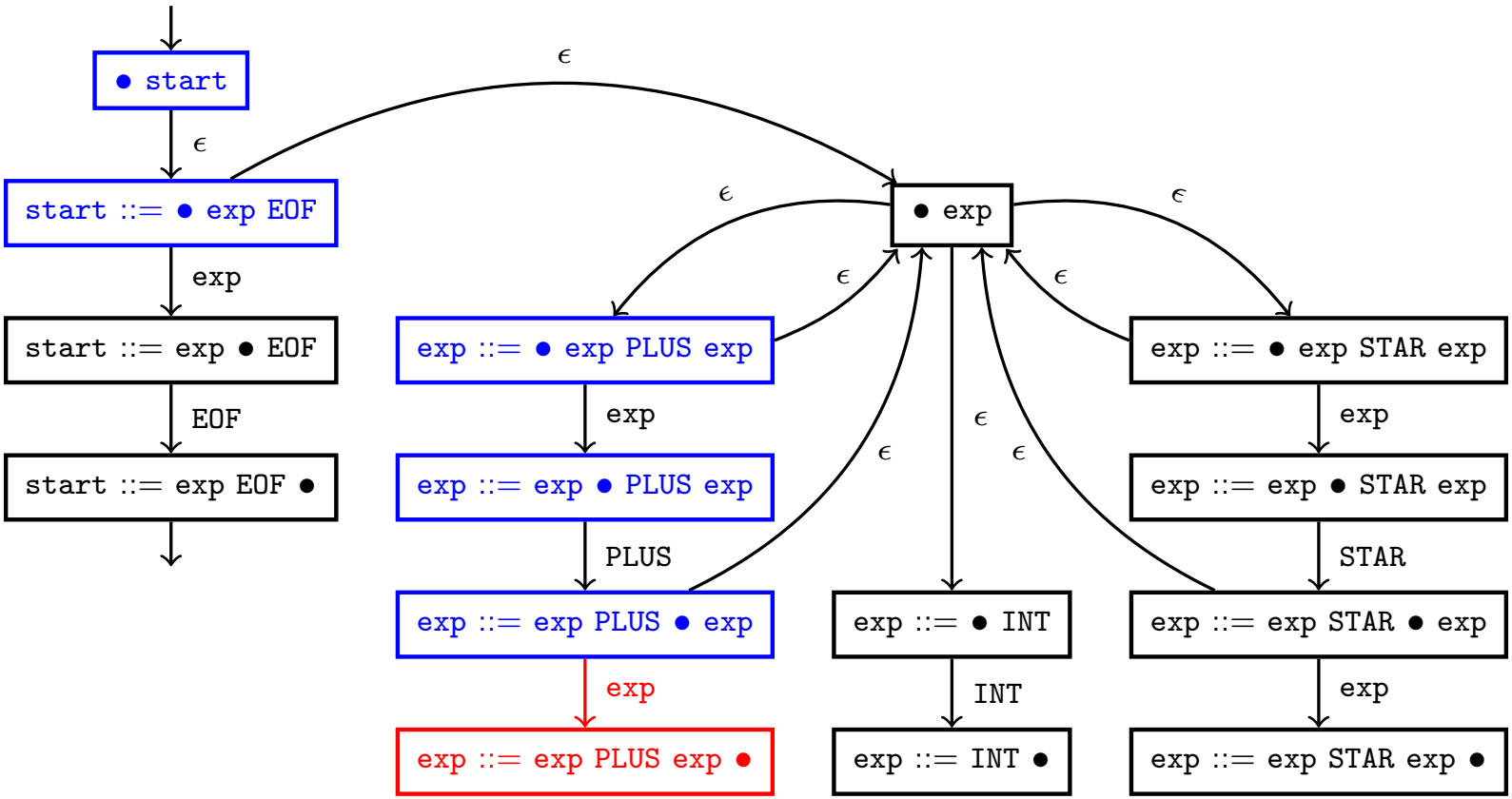
EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$

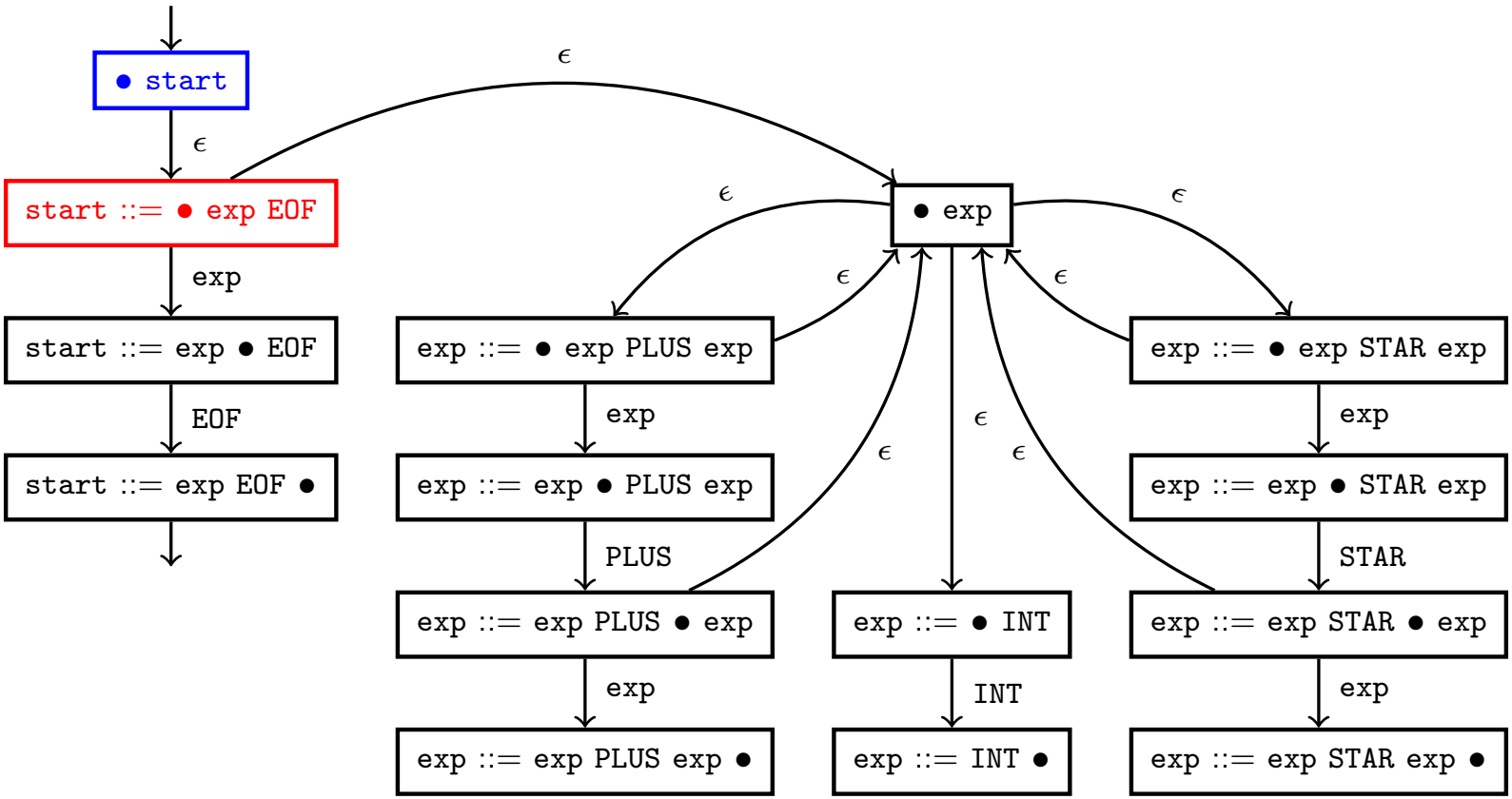


Réduire exp EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$

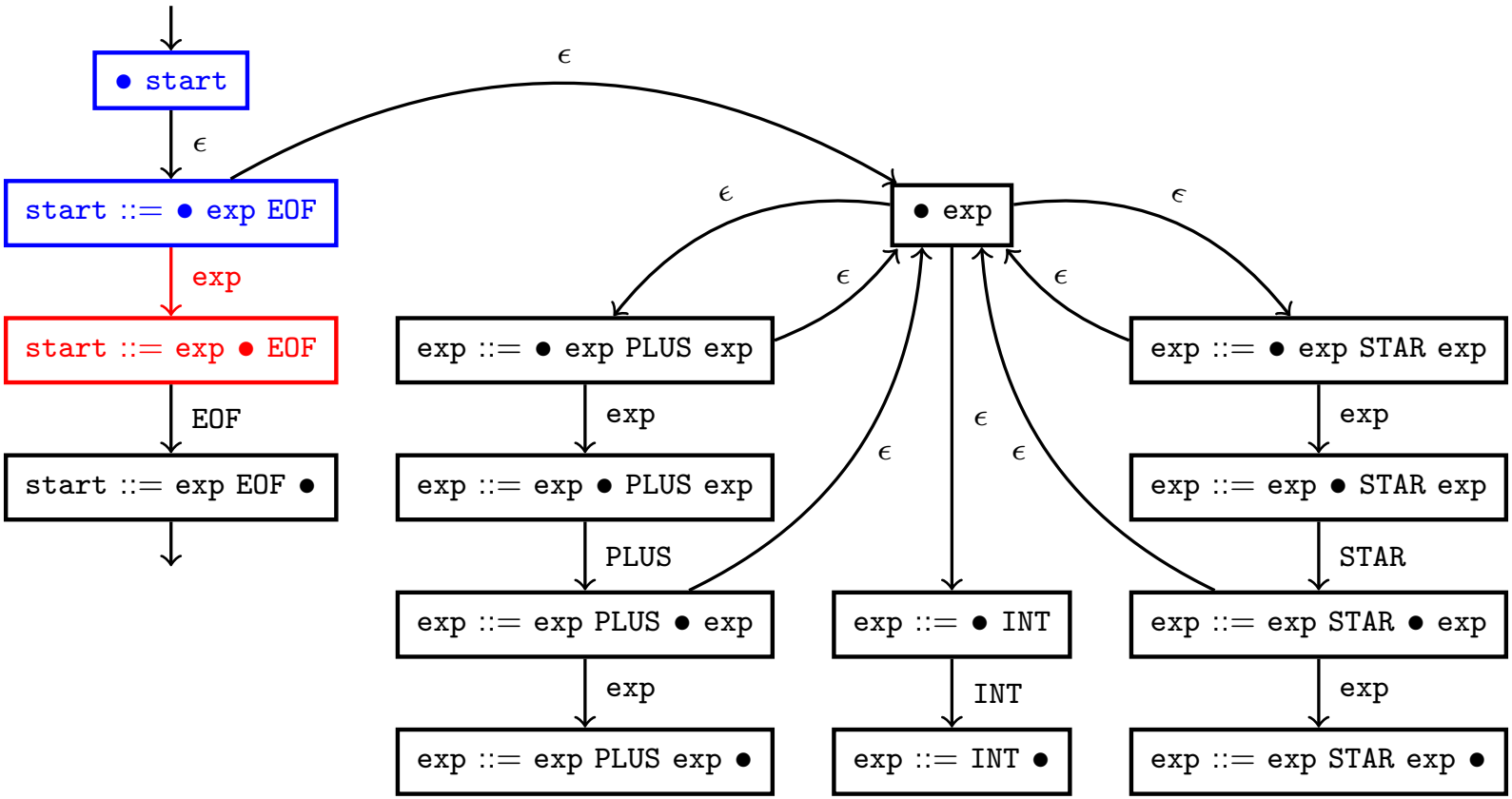


Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



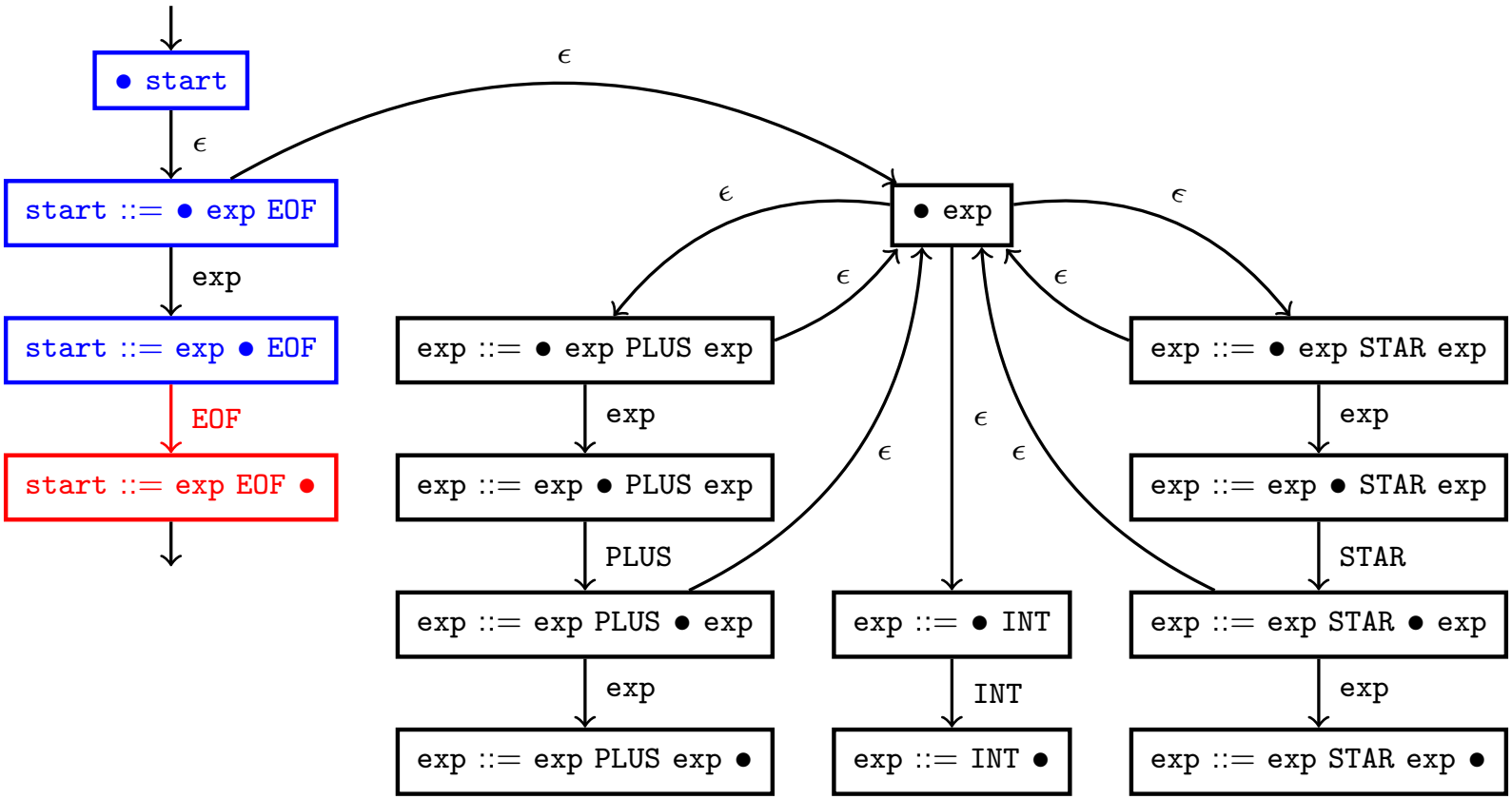
Réduire `exp EOF`

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



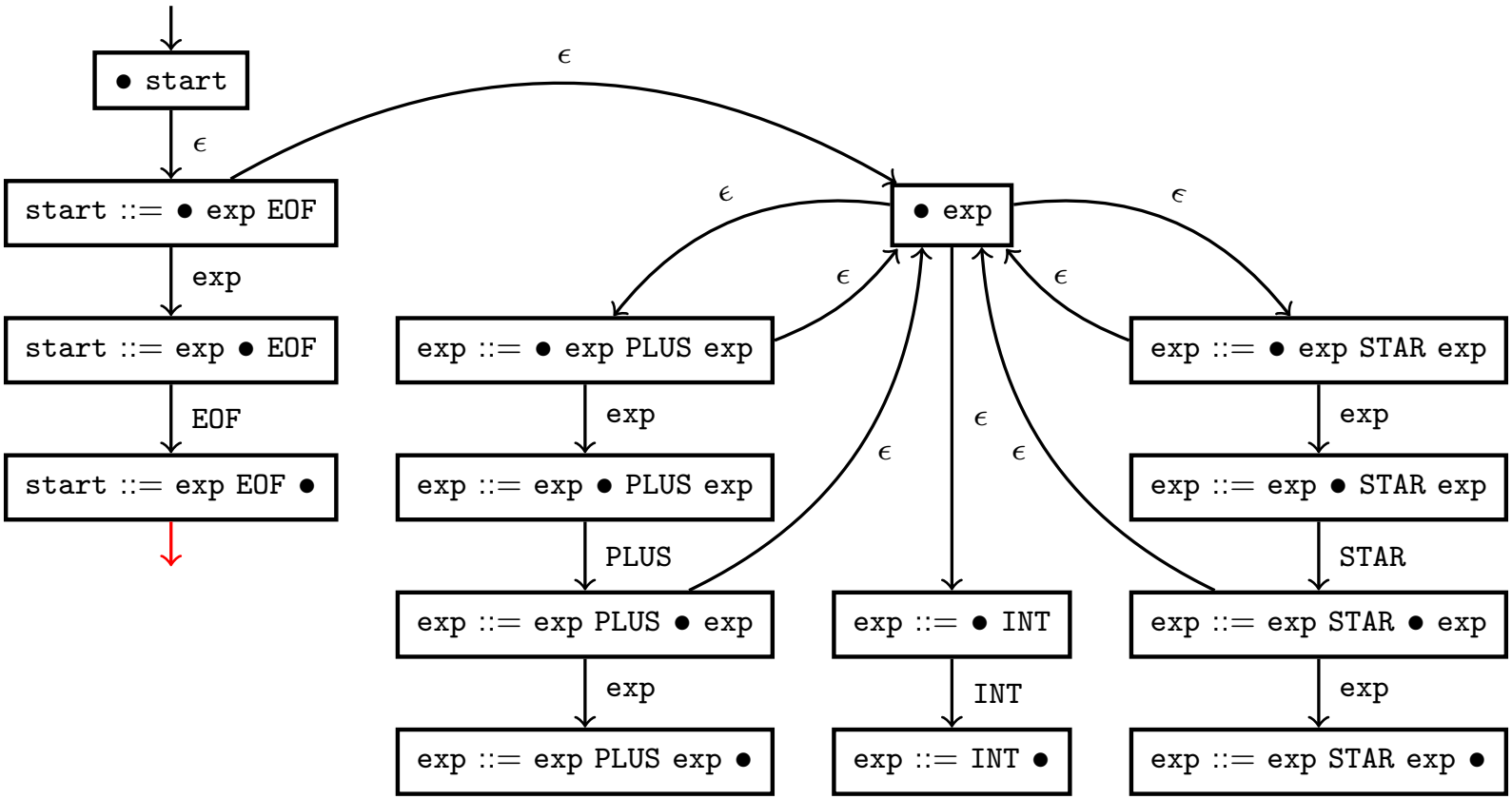
Décaler EOF

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$



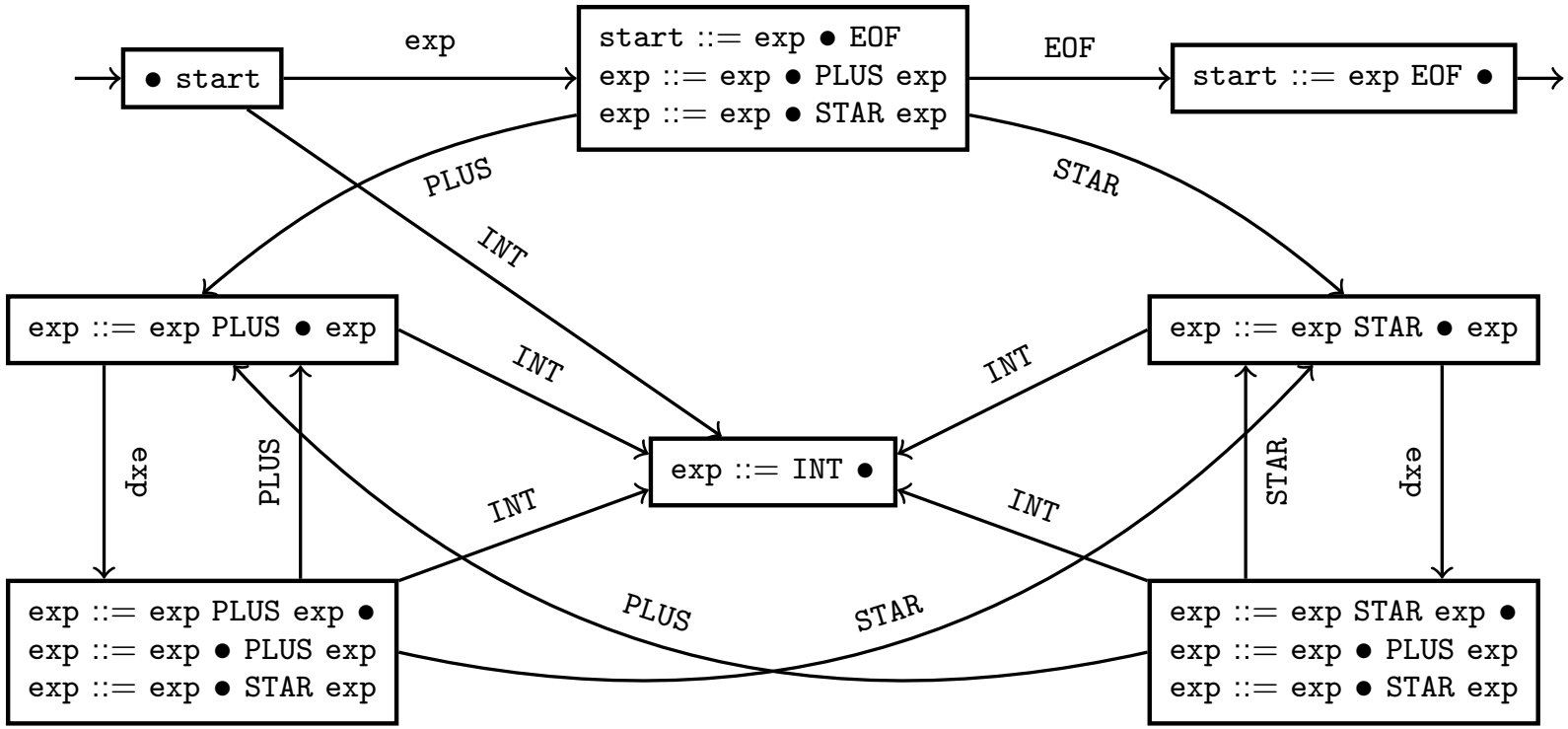
Décaler

Un automate à pile non-déterministe pour $\mathcal{G}_{exp}$

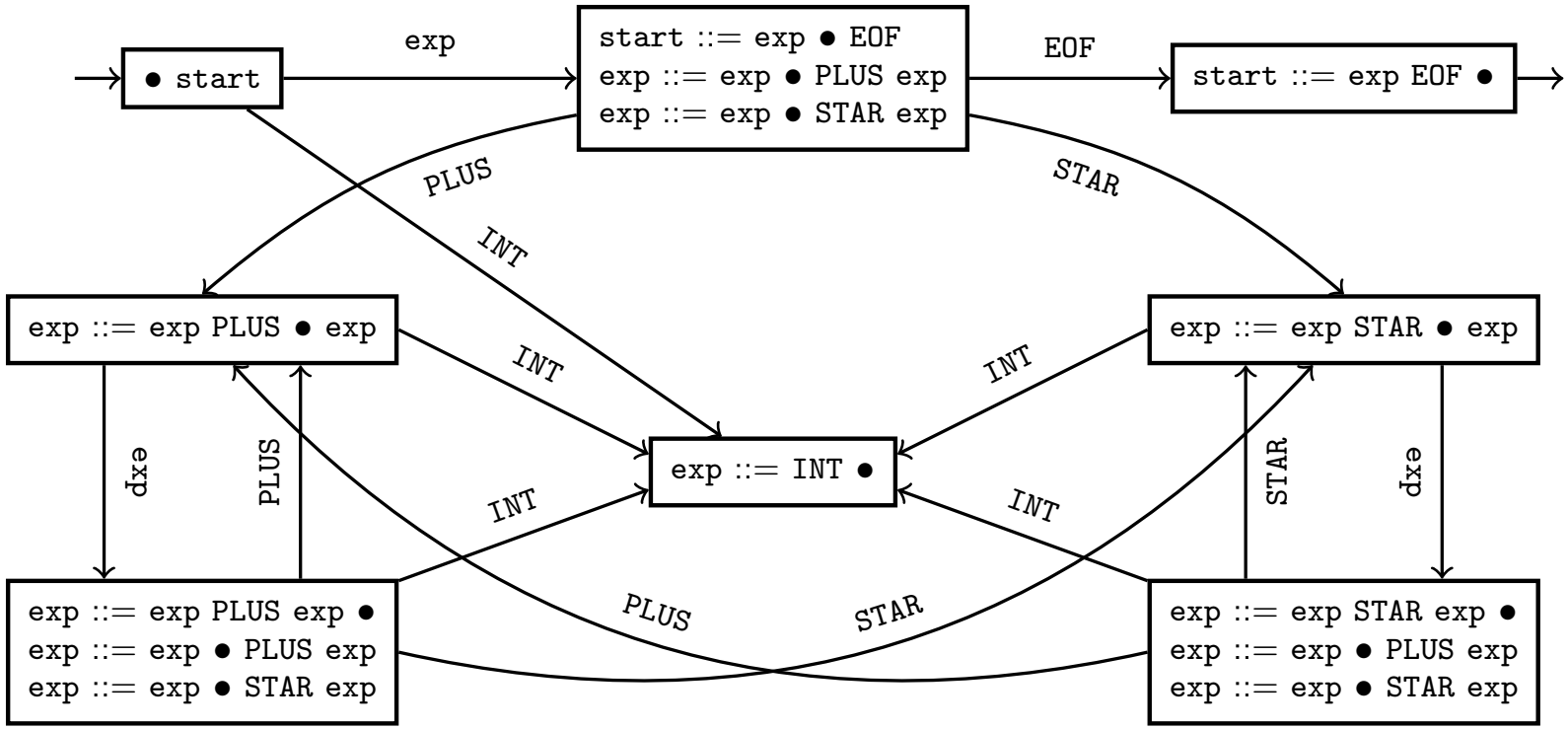


Accepter

Peut-on déterminiser l'automate ?

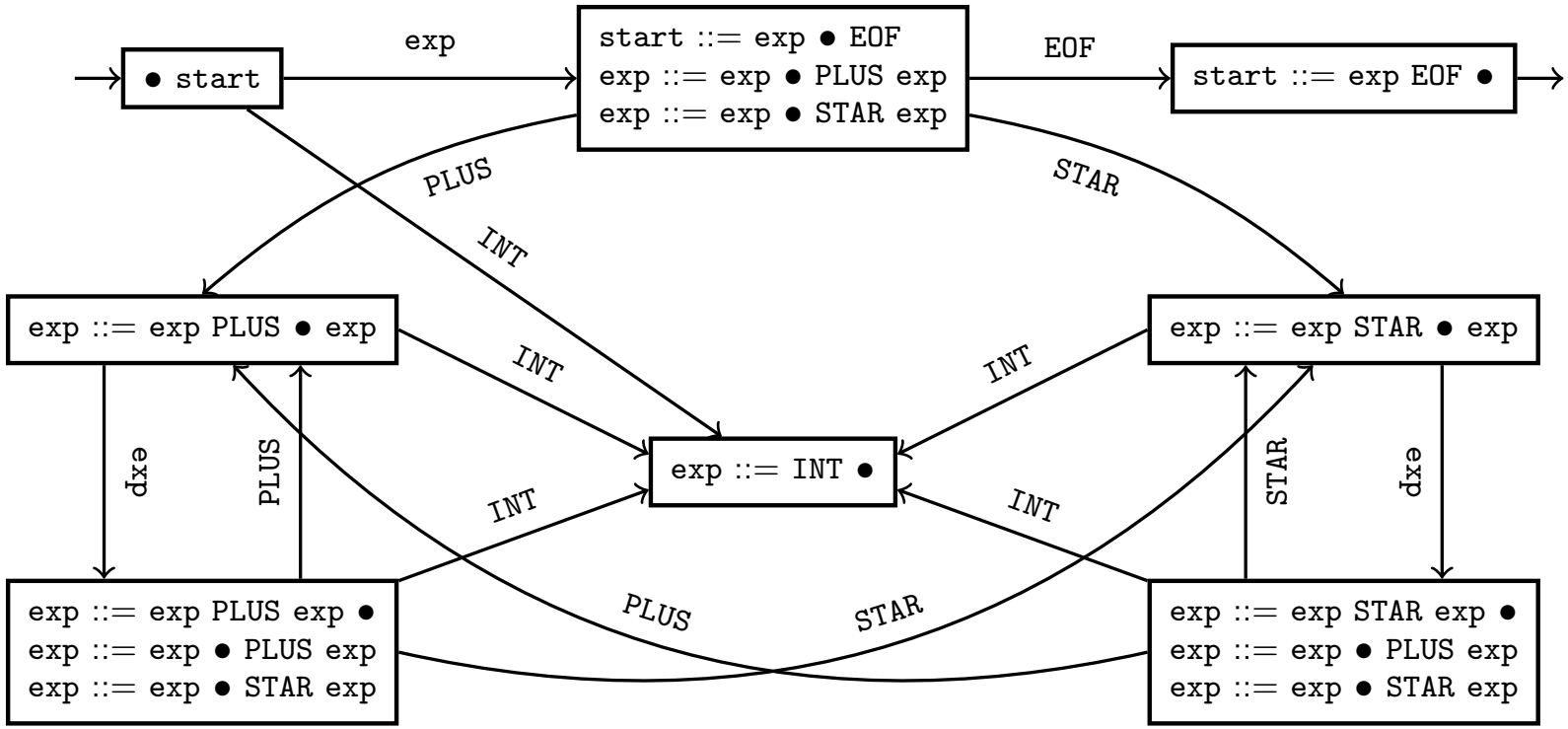


Peut-on déterminiser l'automate ?



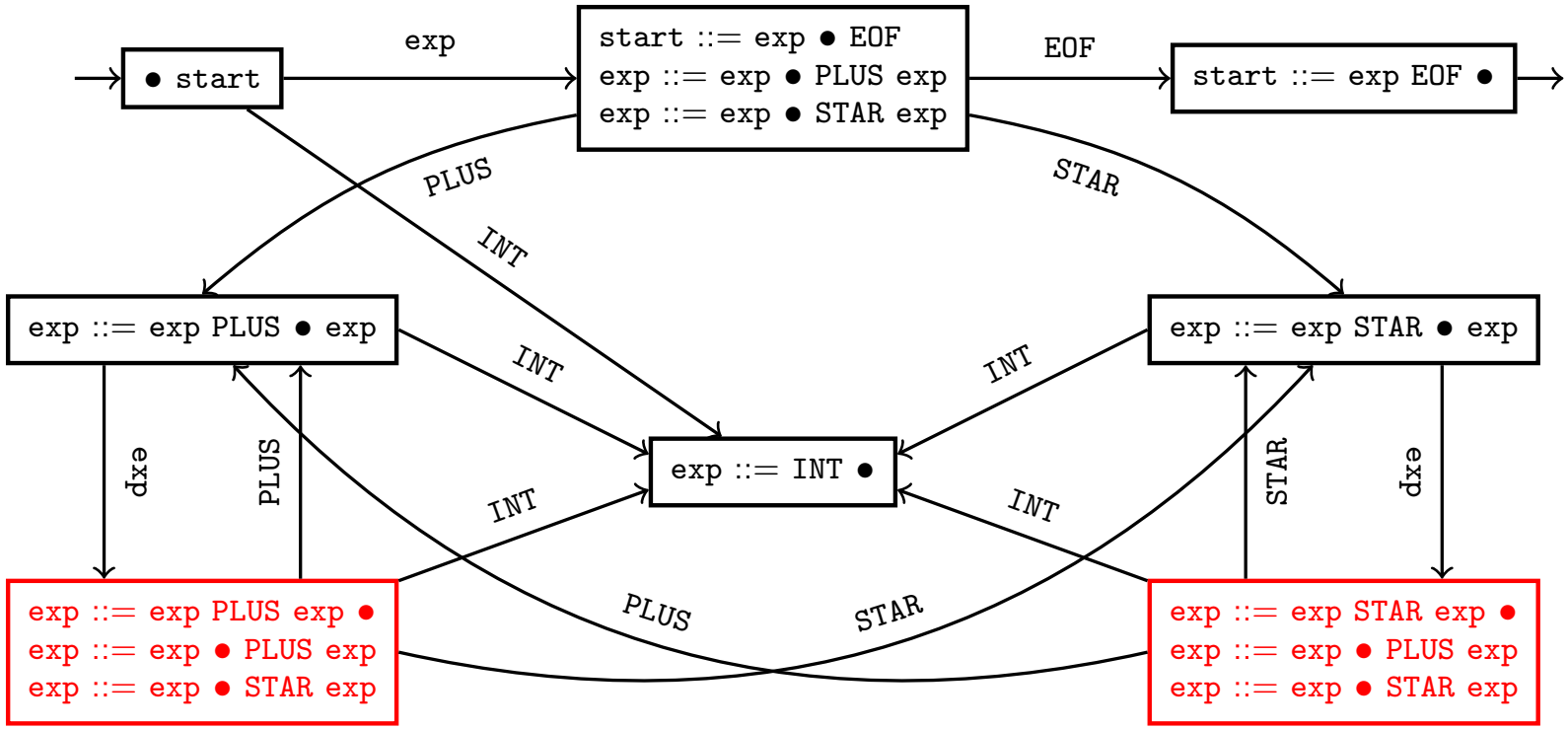
- Cet automate s'appelle l'*automate* $LR(0)$ de $\mathcal{G}_{exp}$.

Peut-on déterminiser l'automate ?



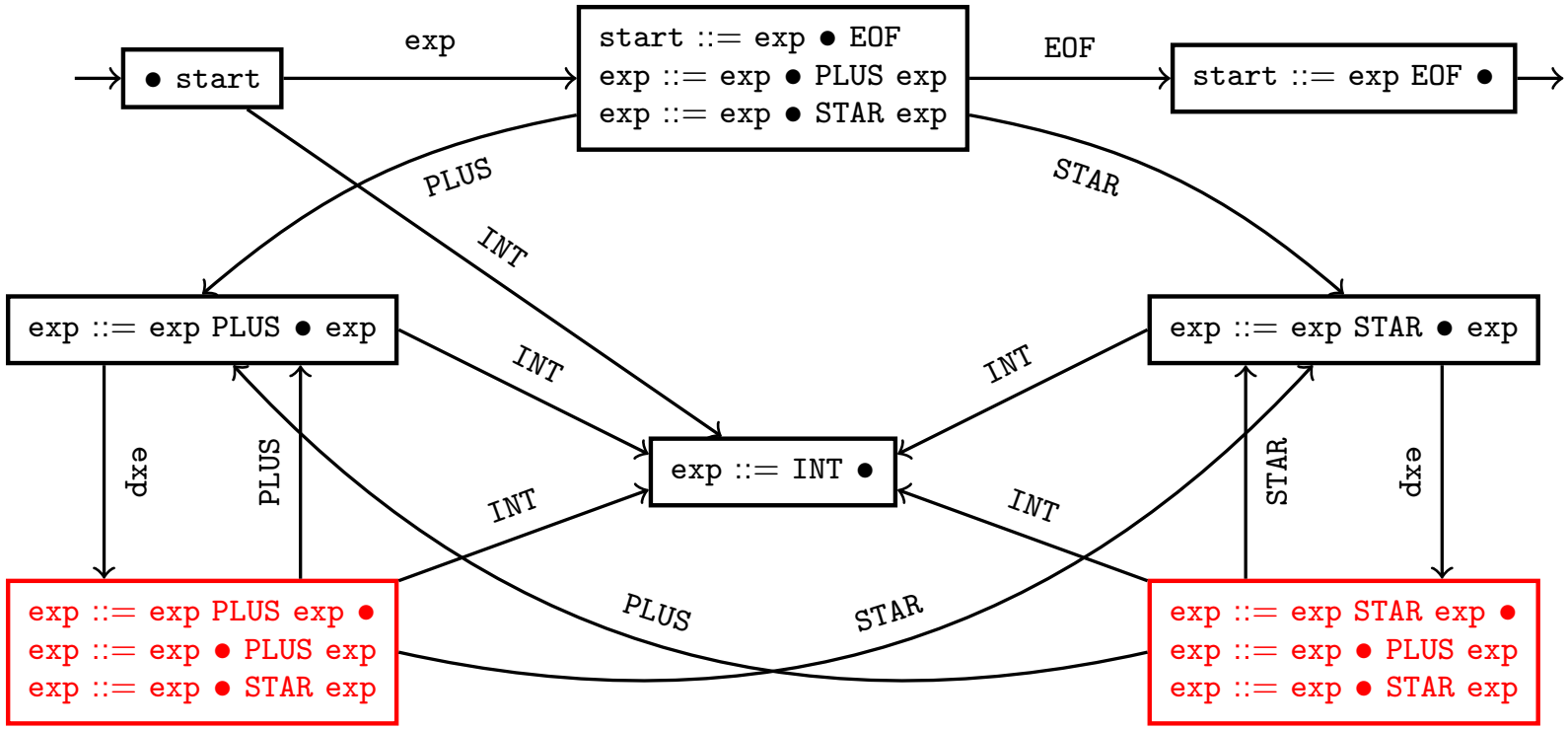
- Cet automate s'appelle l'*automate* $LR(0)$ de $\mathcal{G}_{exp}$.
- Est-il déterministe ?

Peut-on déterminiser l'automate ?



- Cet automate s'appelle l'*automate* $LR(0)$ de $\mathcal{G}_{exp}$.
- Est-il déterministe ? En fait, non : conflits *décaler/réduire*.

Peut-on déterminiser l'automate ?



- Cet automate s'appelle l'*automate* $LR(0)$ de $\mathcal{G}_{exp}$.
- Est-il déterministe ? En fait, non : conflits *décaler/réduire*.
- Par conséquent, on dira que cette grammaire n'est **pas** $LR(0)$.

L'analyse ascendante, en général

Les analyses $LR(k)$

- L'analyse $LR(0)$ utilise uniquement l'état courant de pour décider de la prochaine action à effectuer. On lit l'entrée *a posteriori*.
- Dans l'analyse $LR(k)$, on utilise en plus les k prochaines entrées.
  - Cela nous mène à des automates plus gros et souvent sans conflits.
  - La grammaire est $LR(k)$ si son automate $LR(k)$ est sans conflits. Cela prouve également qu'elle n'est pas ambigüe.
  - Une grammaire ambigüe n'est $LR(k)$ pour aucun k.
- Les analyses $LR(k)$ sont des analyses...
  - *Leftmost* : elles lisent l'entrée de gauche à droite,
  - *Rightward* : elles construisent des dérivations droite,
  - *ascendantes* : on construit la dérivation de bas en haut.

L'analyse ascendante, en général

Les analyses $LR(k)$

- L'analyse $LR(0)$ utilise uniquement l'état courant de pour décider de la prochaine action à effectuer. On lit l'entrée *a posteriori*.
- Dans l'analyse $LR(k)$, on utilise en plus les k prochaines entrées.
  - Cela nous mène à des automates plus gros et souvent sans conflits.
  - La grammaire est $LR(k)$ si son automate $LR(k)$ est sans conflits. Cela prouve également qu'elle n'est pas ambigüe.
  - Une grammaire ambigüe n'est $LR(k)$ pour aucun k.
- Les analyses $LR(k)$ sont des analyses...
  - *Leftmost* : elles lisent l'entrée de gauche à droite,
  - *Rightward* : elles construisent des dérivations droite,
  - *ascendantes* : on construit la dérivation de bas en haut.

Un compromis : l'analyse $LR(1)$

- En pratique, les outils comme Menhir utilisent la classe $LR(1)$.
- Ils signalent les conflits décaler/réduire et réduire/réduire.

Outline

Menhir en pratique

Contenu d'un fichier Menhir

- Des déclarations de non-terminaux typés.
- D'éventuelles déclarations de précédence et associativité.
- Une grammaire dont chaque règle spécifie une *action sémantique*.
  - Les actions sont des expressions OCaml de même type (cf. `ocamllex`).
  - Dans un compilateur, les actions vont produire des arbres de syntaxe.
- D'autres choses encore.
  - Lire `http://gallium.inria.fr/~fpottier/menhir/manual.pdf`.

Menhir en pratique

Contenu d'un fichier Menhir

- Des déclarations de non-terminaux typés.
- D'éventuelles déclarations de précédence et associativité.
- Une grammaire dont chaque règle spécifie une *action sémantique*.
  - Les actions sont des expressions OCaml de même type (cf. `ocamllex`).
  - Dans un compilateur, les actions vont produire des arbres de syntaxe.
- D'autres choses encore.
  - Lire `http://gallium.inria.fr/~fpottier/menhir/manual.pdf`.

La tâche de Menhir

- Traduire le fichier en automate $LR(1)$ implémenté en OCaml.
  - (Menhir, comme `ocamllex`, est donc lui-même un compilateur !)
- Signaler et *expliquer* les conflits décaler/réduire et réduire/réduire.
  - Pensez à bien passer l'option `--explain` à `menhir`.
  - Les explications se trouvent dans le fichier d'extension `.conflicts`.

Un des conflits décaler/réduire de $\mathcal{G}_{exp}$

```
$ menhir --explain parser.mly
Warning: 2 states have shift/reduce conflicts.
Warning: 4 shift/reduce conflicts were arbitrarily resolved.
$ cat parser.conflicts
** Conflict (shift/reduce) in state 7.
** Tokens involved: STAR PLUS
** The following explanations concentrate on token STAR.
** This state is reached from phrase after reading:

expression PLUS expression

** In state 7, looking ahead at STAR, reducing production
** expression -> expression PLUS expression
** is permitted because of the following sub-derivation:

expression STAR expression // lookahead token appears
expression PLUS expression .

** In state 7, looking ahead at STAR, shifting is permitted
** because of the following sub-derivation:

expression PLUS expression
               expression . STAR expression
```

Comment résoudre les conflits avec Menhir ?

Deux règles d'or méthodologiques, à suivre impérativement

- Il faut **corriger les conflits au fur et à mesure**.
- Il faut **corriger les conflits un par un**.

Comment résoudre les conflits avec Menhir ?

Deux règles d'or méthodologiques, à suivre impérativement

- Il faut **corriger les conflits au fur et à mesure**.
- Il faut **corriger les conflits un par un**.

Corriger les conflits

Il y a deux familles de solutions pour supprimer les conflits :

- assigner des priorités/associativités aux non-terminaux et aux règles,
- réécrire la grammaire pour la rendre $LR(1)$.

La seconde est plus générale et compréhensible, et doit donc être préférée.

Corriger les conflits via l'assignation de priorités

```
%token EOF PLUS STAR
%token<int> INT
%start<Ast.e> start

(* Priorités (croissantes) et associativité des opérateurs. *)
%left PLUS
%left STAR

%%

start: e = expression EOF { e }

expression:
| x = INT                                    { EInt x }
| lhs = expression PLUS rhs = expression { EPlus (lhs, rhs) }
| lhs = expression STAR rhs = expression { EMult (lhs, rhs) }
```

Corriger les conflits en restructurant la grammaire

- À quoi ressemble une grammaire $LR(1)$ équivalente à $\mathcal{G}_{exp}$?
  - C'est essentiellement la grammaire donnée dans `marthe.ml` !
- La solution avec les priorités est à préférer dans les cas simples.
  - Notamment pour supprimer les conflits entre opérateurs infixes.
- La restructuration est plus générale, et nécessaire (cf. jalon 1).