Analyse flots de données et introduction à la forme SSA

Adrien Guatto et Yann Régis-Gianas

IRIF & Université Denis Diderot

Compilation - M1 2018/2019

Où en sommes-nous ?

Les semaines précédentes, vous avez étudié l'**allocation de registres**.

C'est une optimisation très importante :

- Elle améliore très significativement les performances.
- Sa présence contribue à simplifier le reste du compilateur.
- Elle met en jeu votre première analyse du flot des données (*dataflow*).

Où en sommes-nous ?

Les semaines précédentes, vous avez étudié l'**allocation de registres**.

C'est une optimisation très importante :

- Elle améliore très significativement les performances.
- Sa présence contribue à simplifier le reste du compilateur.
- Elle met en jeu votre première analyse du flot des données (*dataflow*).

Ce dernier concept est à la source de nombreux progrès en compilation optimisante depuis les années 1970.

Où en sommes-nous ?

Les semaines précédentes, vous avez étudié l'**allocation de registres**.

C'est une optimisation très importante :

- Elle améliore très significativement les performances.
- Sa présence contribue à simplifier le reste du compilateur.
- Elle met en jeu votre première analyse du flot des données (*dataflow*).

Ce dernier concept est à la source de nombreux progrès en compilation optimisante depuis les années 1970.

La séance d'aujourd'hui

- D'autres analyses du flot de données.
- Leur impact sur l'architecture des compilateurs via la forme SSA.

Où en sommes-nous ?

Les semaines précédentes, vous avez étudié l'**allocation de registres**.

C'est une optimisation très importante :

- Elle améliore très significativement les performances.
- Sa présence contribue à simplifier le reste du compilateur.
- Elle met en jeu votre première analyse du flot des données (*dataflow*).

Ce dernier concept est à la source de nombreux progrès en compilation optimisante depuis les années 1970.

La séance d'aujourd'hui

- D'autres analyses du flot de données.
- Leur impact sur l'architecture des compilateurs via la forme SSA.

(Pas de jalon correspondant, détendez-vous.)

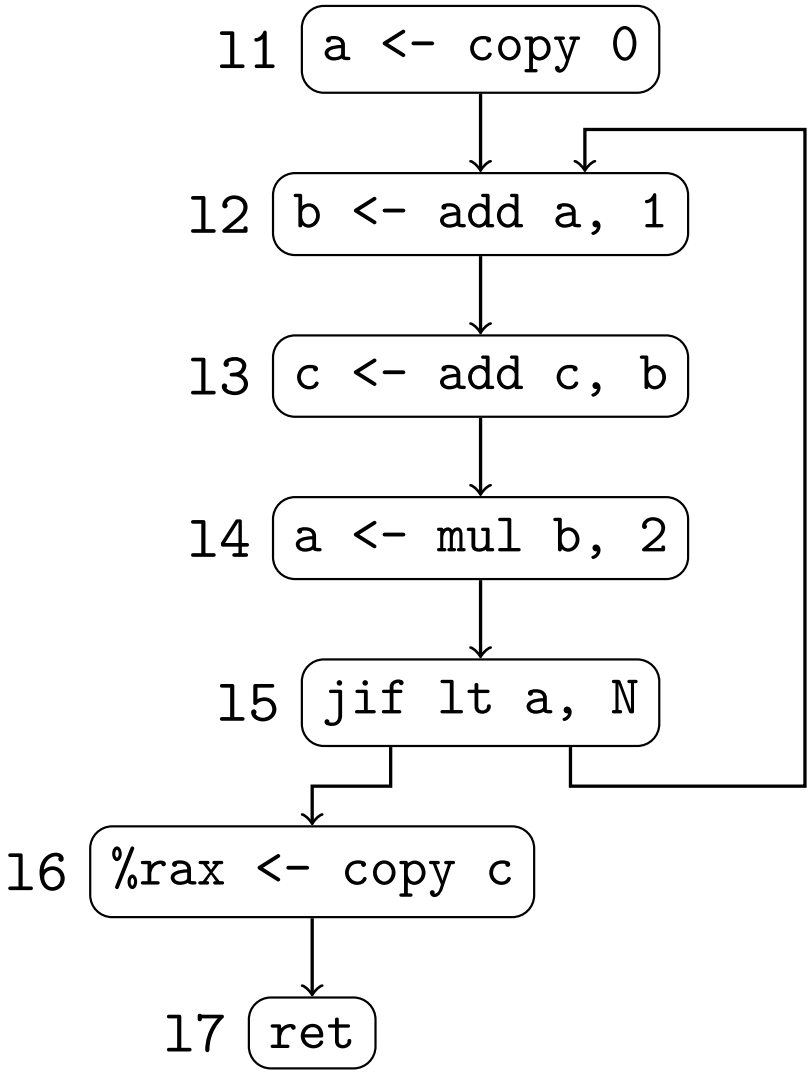
Outline

Retour sur l'analyse des durées de vie

```
l1: a <- copy 0;
l2: b <- add a, 1;
l3: c <- add c, b;
l4: a <- mul b, 2;
l5: jif lt a, N -> l2, l6;
l6: %rax <- copy c;
l7: ret;
```

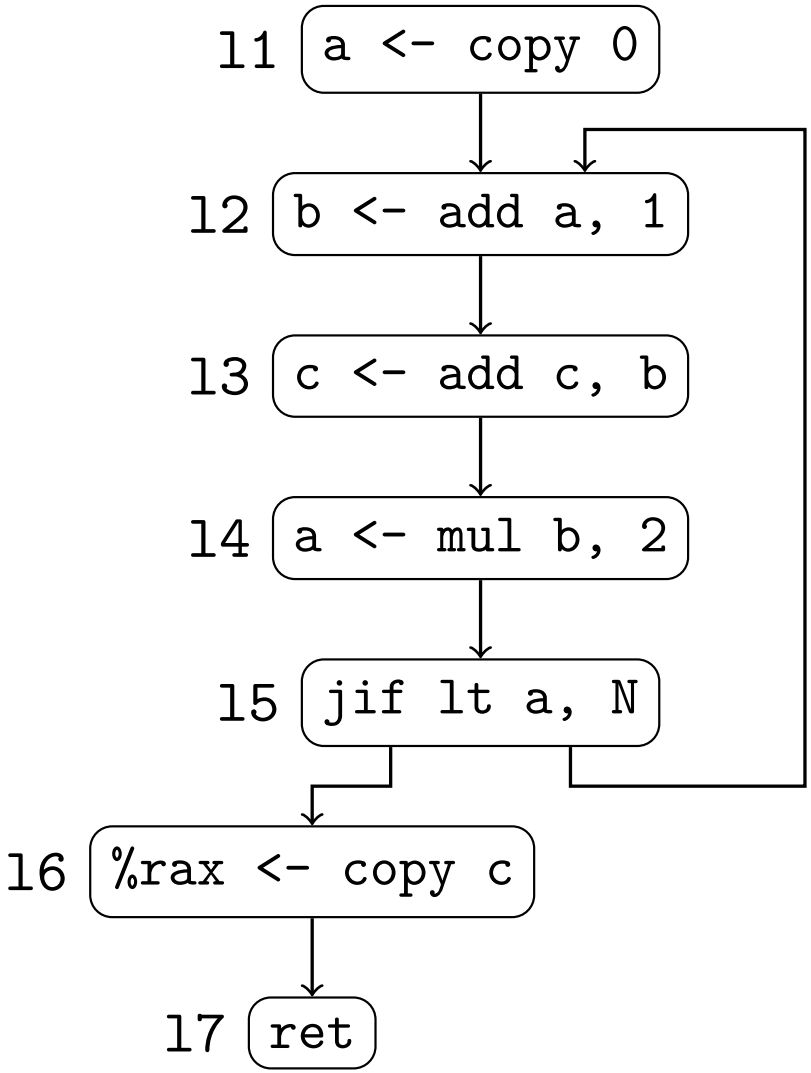
Retour sur l'analyse des durées de vie

```
l1: a <- copy 0;
l2: b <- add a, 1;
l3: c <- add c, b;
l4: a <- mul b, 2;
l5: jif lt a, N -> l2, l6;
l6: %rax <- copy c;
l7: ret;
```



Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

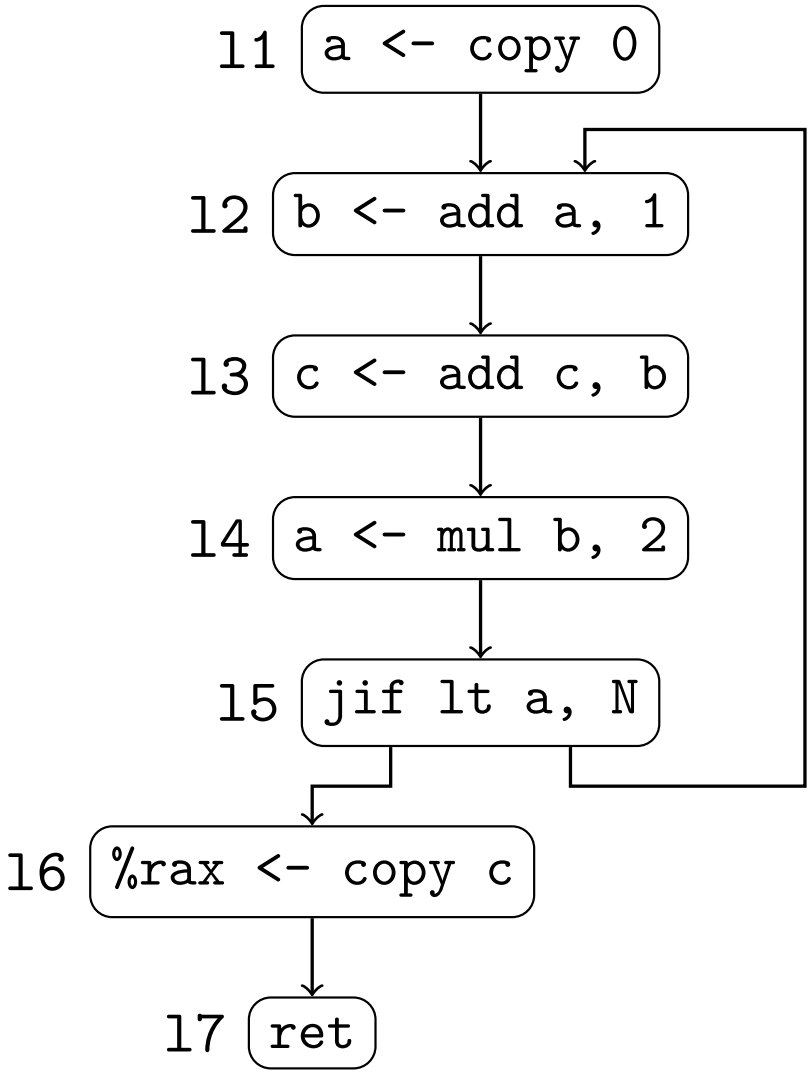


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$

$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
l1	$\emptyset$	$\{a\}$	$\emptyset$	$\emptyset$
l2	$\{a\}$	$\{b\}$	$\emptyset$	$\emptyset$
l3	$\{b, c\}$	$\{c\}$	$\emptyset$	$\emptyset$
l4	$\{b\}$	$\{a\}$	$\emptyset$	$\emptyset$
l5	$\{a\}$	$\emptyset$	$\emptyset$	$\emptyset$
l6	$\{c\}$	$\emptyset$	$\emptyset$	$\emptyset$



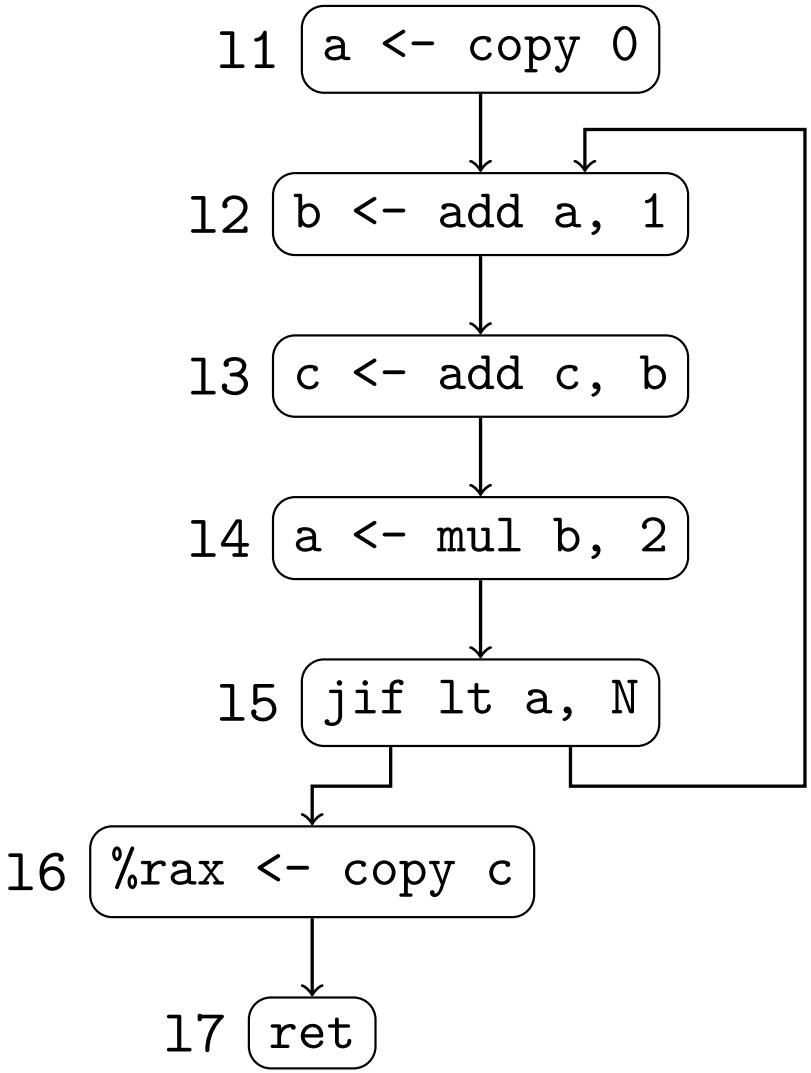
Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$

$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\emptyset$	$\emptyset$
12	$\{a\}$	$\{b\}$	$\emptyset$	$\emptyset$
13	$\{b, c\}$	$\{c\}$	$\emptyset$	$\emptyset$
14	$\{b\}$	$\{a\}$	$\emptyset$	$\emptyset$
15	$\{a\}$	$\emptyset$	$\emptyset$	$\emptyset$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 1.



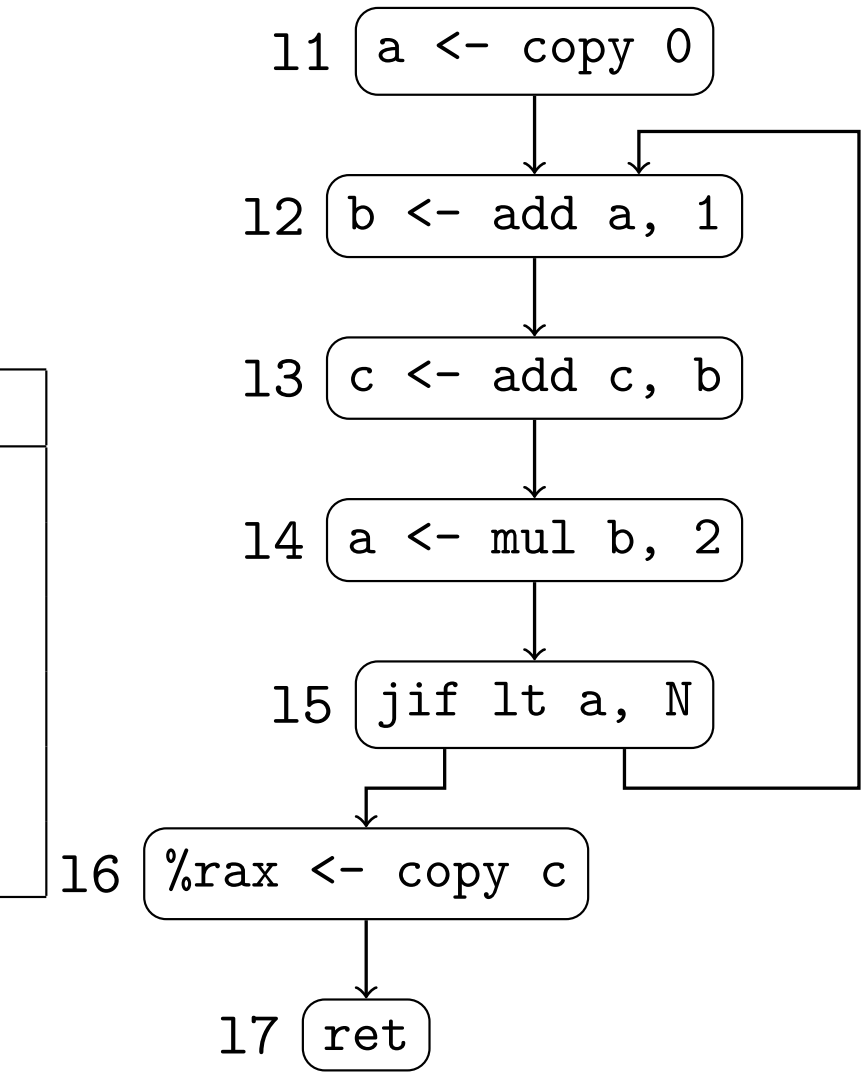
Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$

$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\emptyset$	$\emptyset$
12	$\{a\}$	$\{b\}$	$\emptyset$	$\emptyset$
13	$\{b, c\}$	$\{c\}$	$\emptyset$	$\emptyset$
14	$\{b\}$	$\{a\}$	$\emptyset$	$\emptyset$
15	$\{a\}$	$\emptyset$	$\{c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 1.

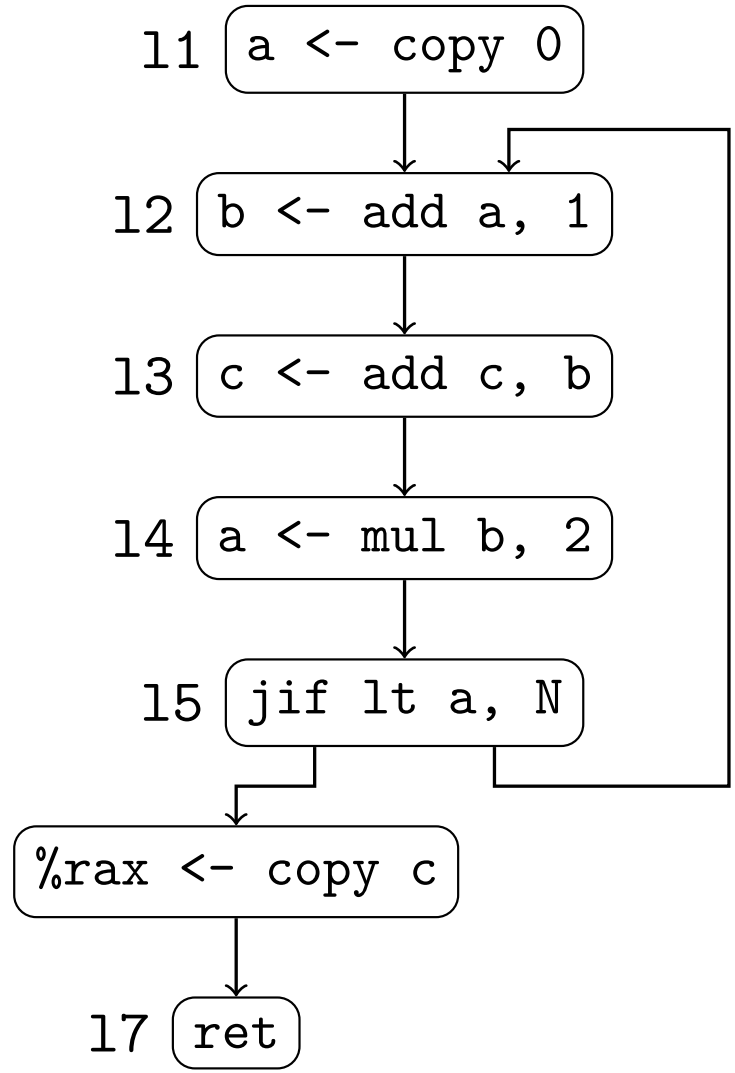


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\emptyset$	$\emptyset$
12	$\{a\}$	$\{b\}$	$\emptyset$	$\emptyset$
13	$\{b, c\}$	$\{c\}$	$\emptyset$	$\emptyset$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 1.

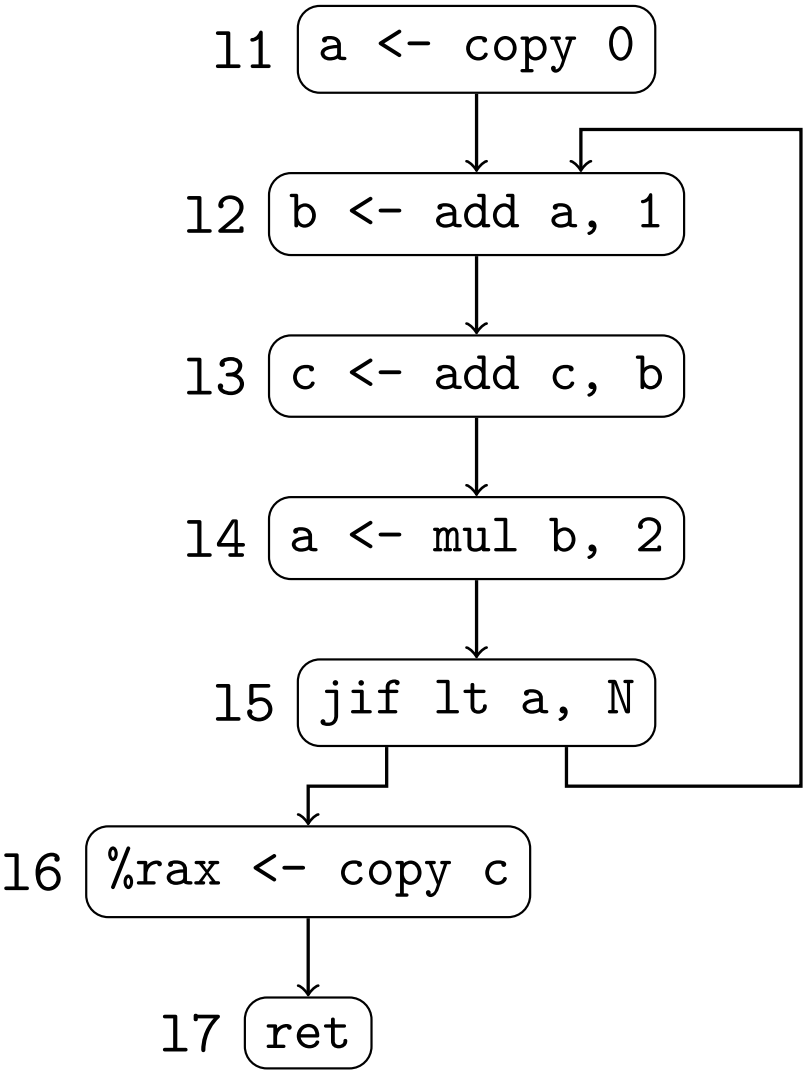


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\emptyset$	$\emptyset$
12	$\{a\}$	$\{b\}$	$\emptyset$	$\emptyset$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 1.

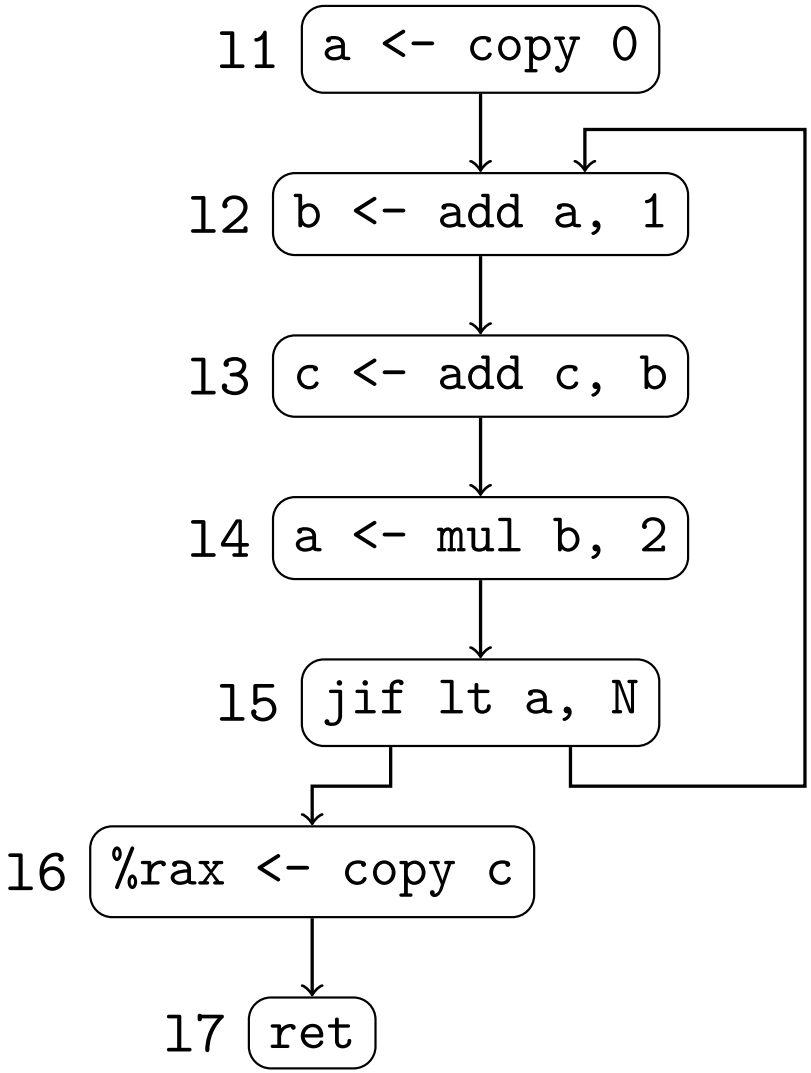


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\emptyset$	$\emptyset$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 1.

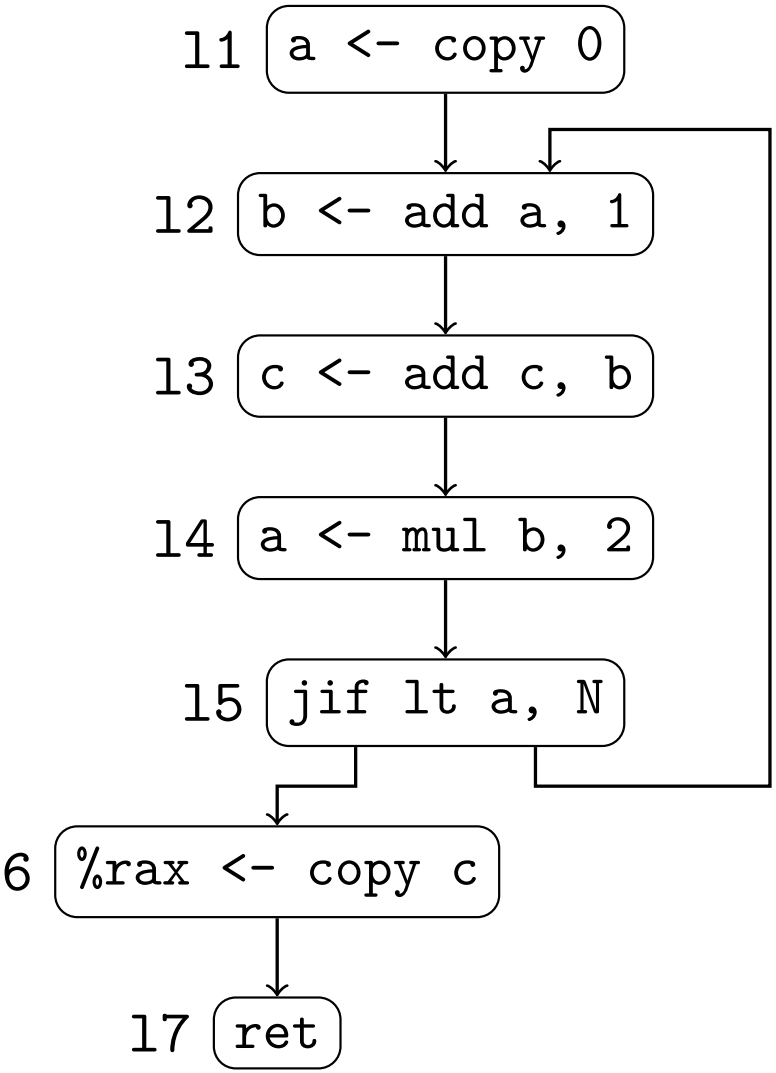


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 1.

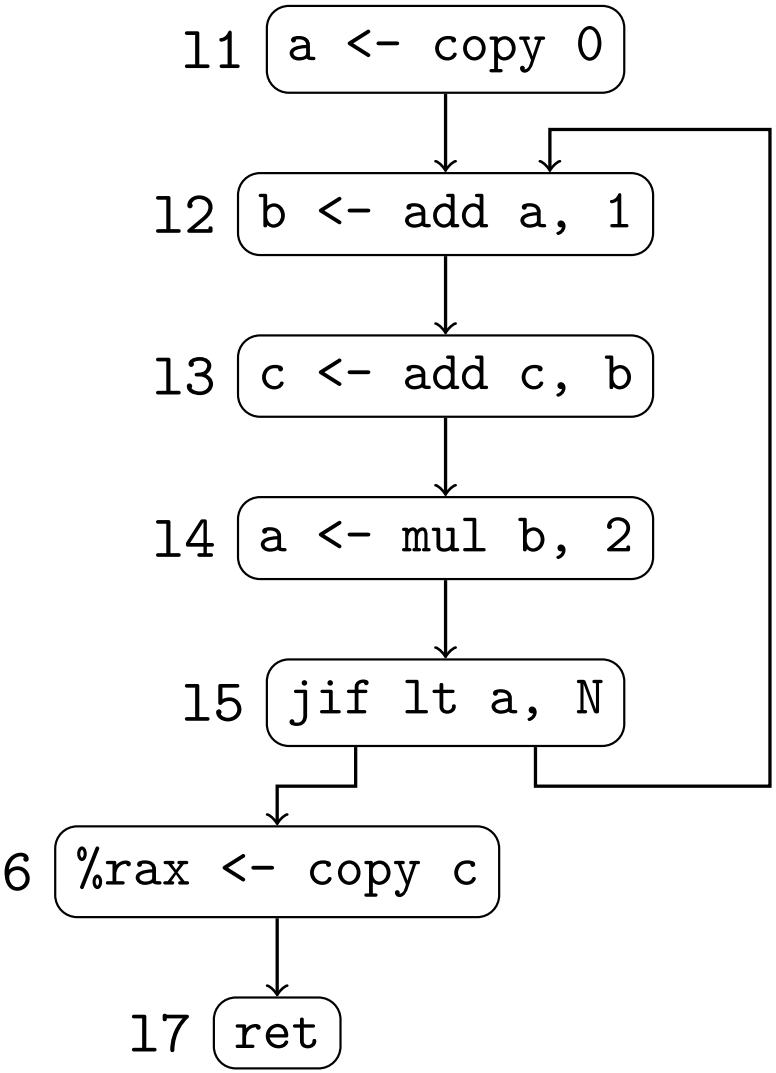


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 2.

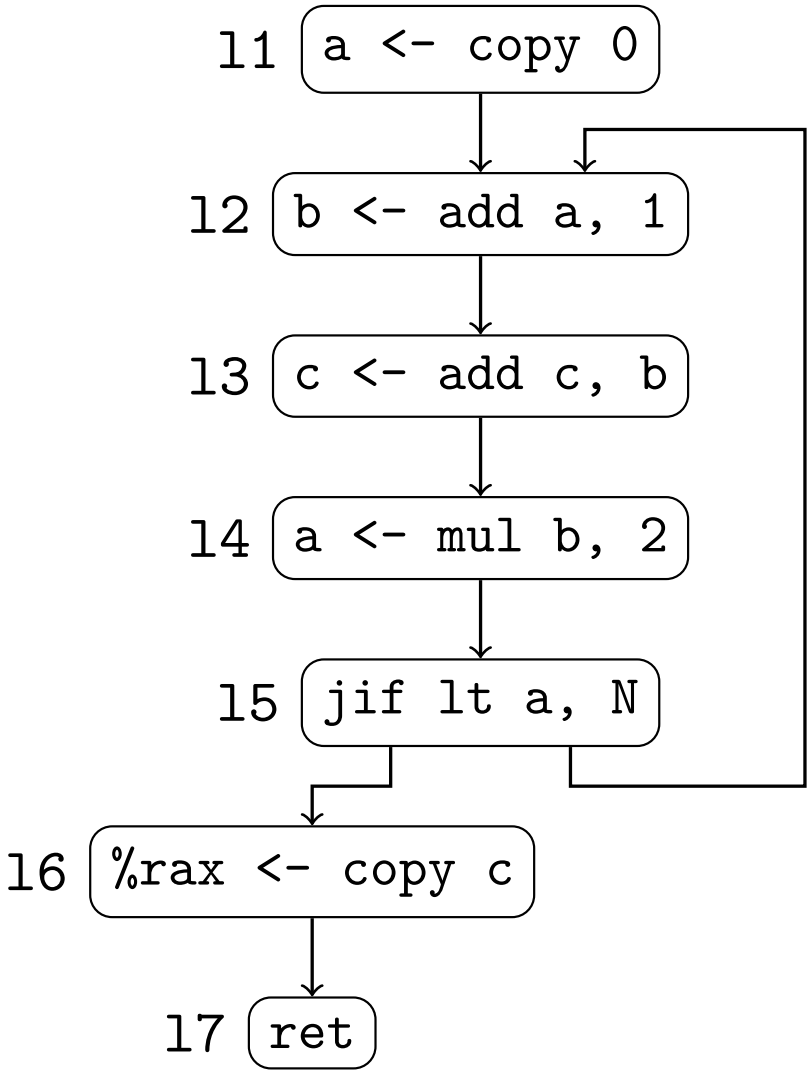


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{a, c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 2.

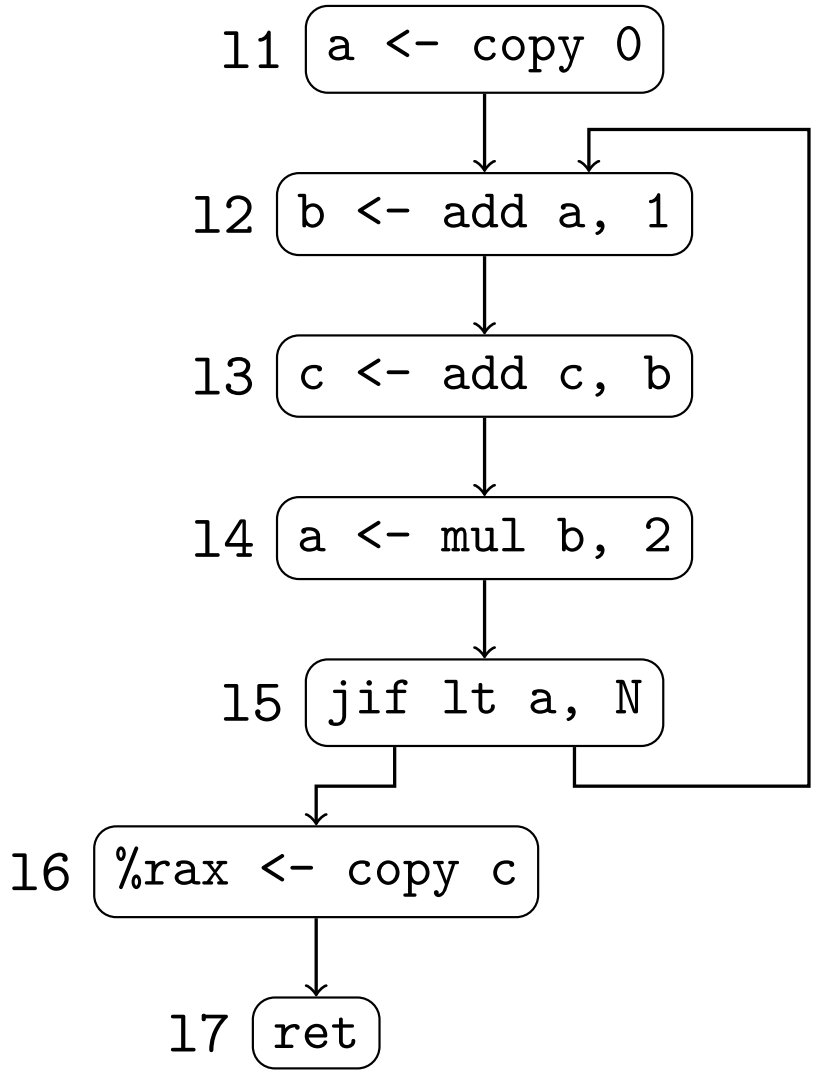


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{a, c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 2.

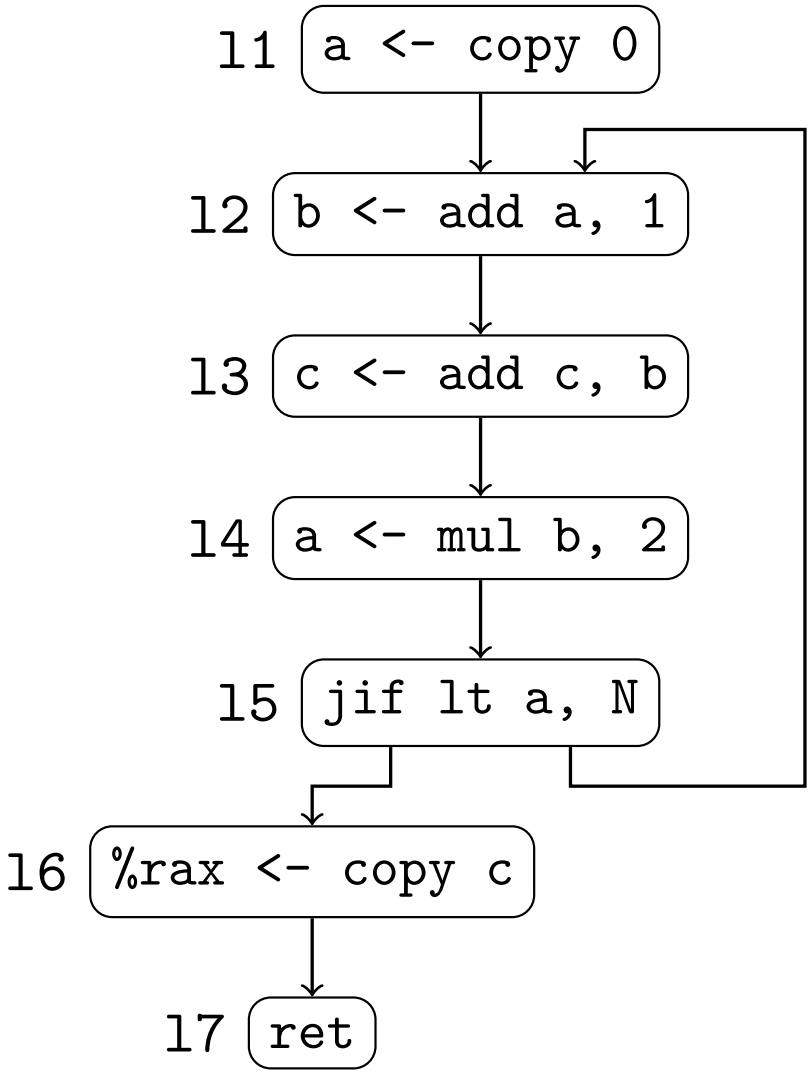


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{a, c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 2.

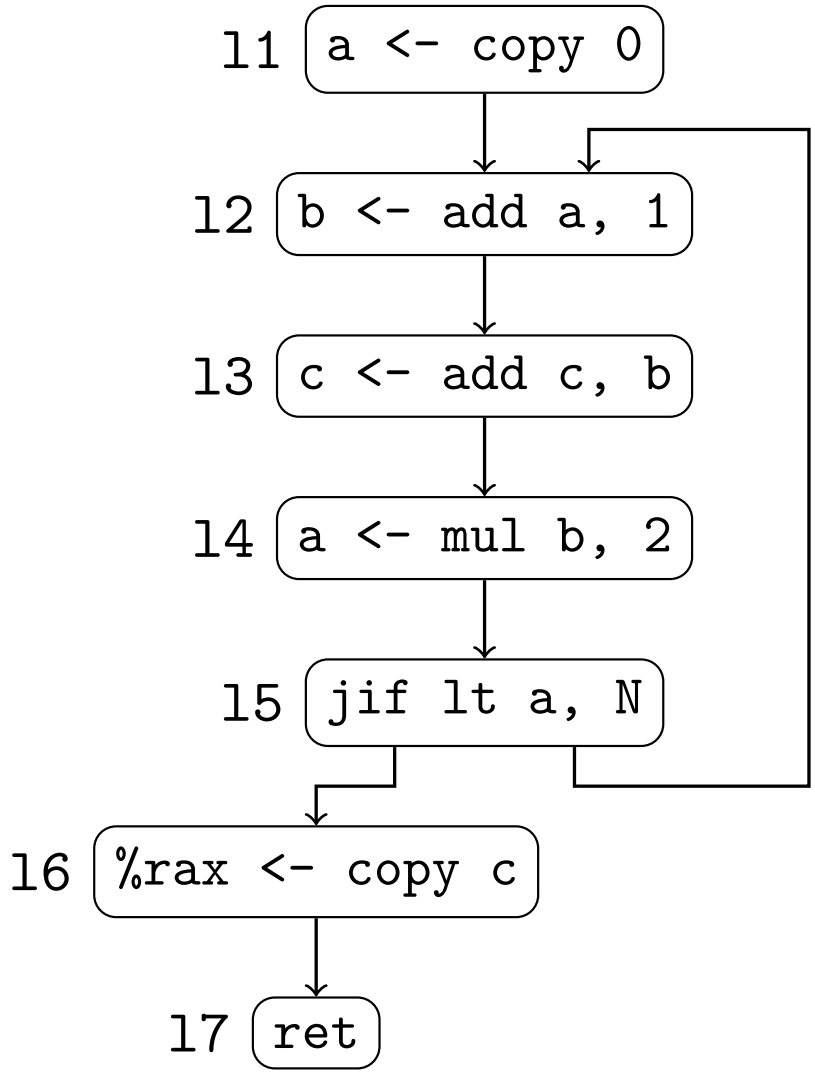


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{a, c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 2.

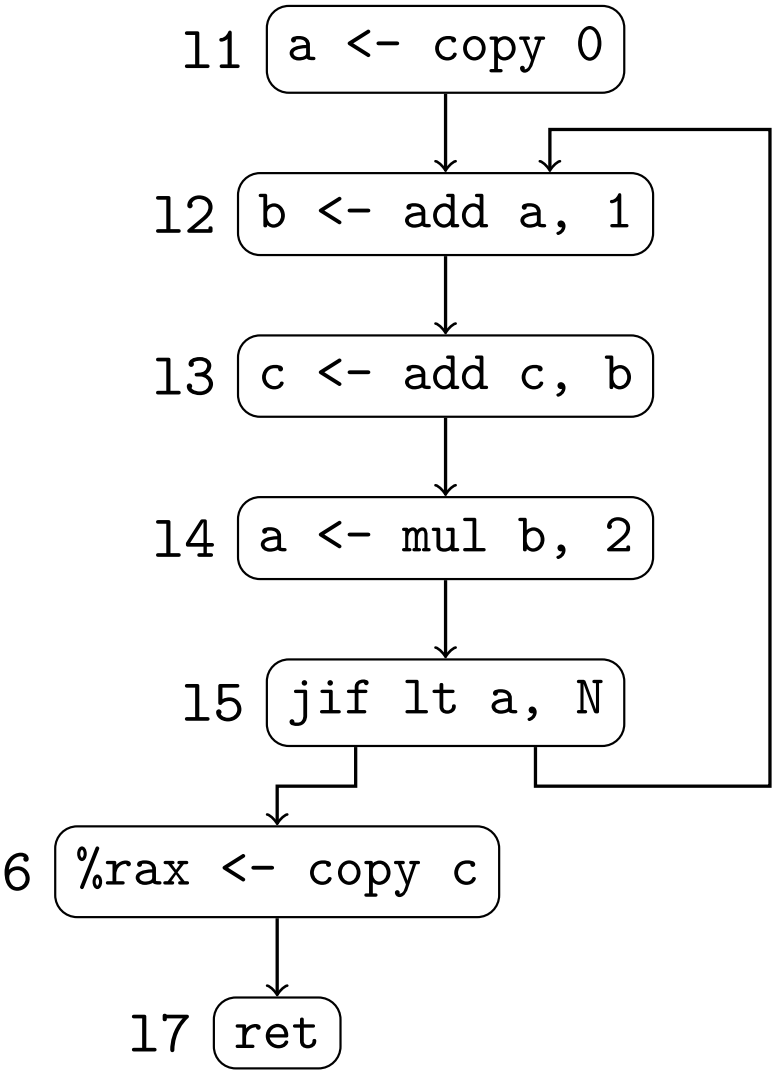


Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{a, c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 2.



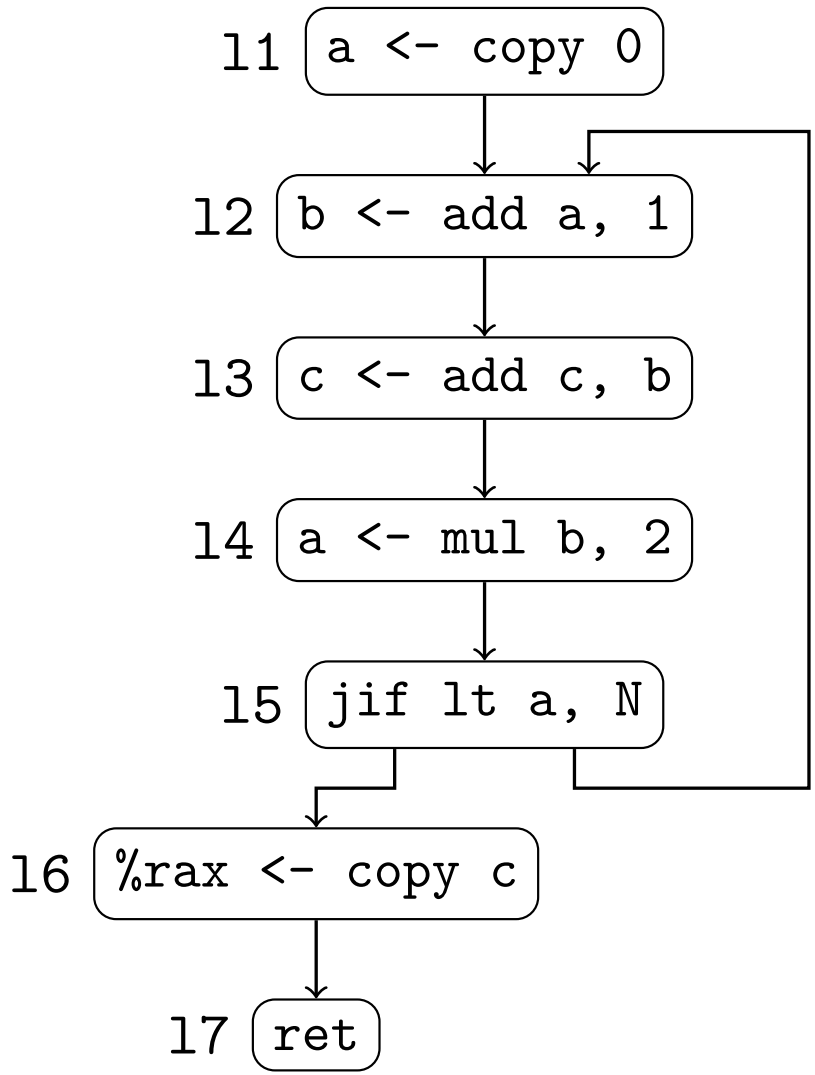
Retour sur l'analyse des durées de vie

$$LIn(l) = Use(l) \cup (LOut(l) \setminus Def(l))$$
$$LOut(l) = \bigcup_{l' \in Succ(l)} LIn(l')$$

l	$Use(l)$	$Def(l)$	$LOut(l)$	$LIn(l)$
11	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$
12	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$
13	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$
14	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$
15	$\{a\}$	$\emptyset$	$\{a, c\}$	$\{a, c\}$
16	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$

Itération 2.

Point fixe atteint.



L'analyse des durées de vie, plus formellement

Le résultat r de l'analyse associe à chaque label $l \in Lab$ un élément de

$$P \triangleq \underbrace{\mathcal{P}(Var)}_{LOut(l)} \times \underbrace{\mathcal{P}(Var)}_{LIn(l)}.$$

L'analyse des durées de vie, plus formellement

Le résultat r de l'analyse associe à chaque label $l \in Lab$ un élément de

$$P \triangleq \underbrace{\mathcal{P}(Var)}_{LOut(l)} \times \underbrace{\mathcal{P}(Var)}_{LIn(l)}.$$

Ce r est la solution d'un système d'équations défini par notre programme.

$$r.\texttt{l1}.LOut = r.\texttt{l2}.LIn \qquad r.\texttt{l2}.LOut = r.\texttt{l3}.LIn$$
$$r.\texttt{l1}.LIn = r.\texttt{l1}.LOut \setminus \{a\} \qquad r.\texttt{l2}.LIn = \{a\} \cup (r.\texttt{l2}.LOut \setminus \{b\}) \quad \cdots$$

L'analyse des durées de vie, plus formellement

Le résultat r de l'analyse associe à chaque label $l \in Lab$ un élément de

$$P \triangleq \underbrace{\mathcal{P}(Var)}_{LOut(l)} \times \underbrace{\mathcal{P}(Var)}_{LIn(l)}.$$

Ce r est la solution d'un système d'équations défini par notre programme.

$$\begin{array}{ll} r.\mathtt{11}.LOut = r.\mathtt{12}.LIn & r.\mathtt{12}.LOut = r.\mathtt{13}.LIn \\ r.\mathtt{11}.LIn = r.\mathtt{11}.LOut \setminus \{a\} & r.\mathtt{12}.LIn = \{a\} \cup (r.\mathtt{12}.LOut \setminus \{b\}) \quad \cdots \end{array}$$

On peut voir ce système d'équations comme une famille de fonctions de transfert $F_l : (Lab \to P) \to P$, une pour chaque label l.

$$\begin{array}{l} F_{\mathtt{11}}(x) = (x.\mathtt{12}.LIn, x.\mathtt{11}.Out \setminus \{a\}) \\ F_{\mathtt{12}}(x) = (x.\mathtt{13}.LIn, \{a\} \cup (x.\mathtt{12}.Out \setminus \{b\})) \end{array} \quad \cdots$$

L'analyse des durées de vie, plus formellement

Le résultat r de l'analyse associe à chaque label $l \in Lab$ un élément de

$$P \triangleq \underbrace{\mathcal{P}(Var)}_{LOut(l)} \times \underbrace{\mathcal{P}(Var)}_{LIn(l)}.$$

Ce r est la solution d'un système d'équations défini par notre programme.

$$\begin{array}{ll} r.\mathtt{11}.LOut = r.\mathtt{12}.LIn & r.\mathtt{12}.LOut = r.\mathtt{13}.LIn \\ r.\mathtt{11}.LIn = r.\mathtt{11}.LOut \setminus \{a\} & r.\mathtt{12}.LIn = \{a\} \cup (r.\mathtt{12}.LOut \setminus \{b\}) \quad \cdots \end{array}$$

On peut voir ce système d'équations comme une famille de fonctions de transfert $F_l : (Lab \to P) \to P$, une pour chaque label l.

$$\begin{array}{l} F_{\mathtt{11}}(x) = (x.\mathtt{12}.LIn, x.\mathtt{11}.Out \setminus \{a\}) \\ F_{\mathtt{12}}(x) = (x.\mathtt{13}.LIn, \{a\} \cup (x.\mathtt{12}.Out \setminus \{b\})) \quad \cdots \end{array}$$

Être une solution du système revient à être un point fixe $r = F(r)$ de :

$$F : (Lab \to P) \to (Lab \to P)$$
$$F(x) = l \mapsto F_l(x)$$

L'analyse des durées de vies, une analyse dataflow

Dans l'exemple, on a trouvé un point fixe en itérant F. Était-ce garanti ?

L'analyse des durées de vies, une analyse dataflow

Dans l'exemple, on a trouvé un point fixe en itérant F. Était-ce garanti ?

Oui ! Pourquoi ?

L'analyse des durées de vies, une analyse dataflow

Dans l'exemple, on a trouvé un point fixe en itérant F. Était-ce garanti ?

Oui ! Pourquoi ?

- P est un ensemble fini ordonné et muni d'un plus petit élément $\bot$.

$$x \sqsubseteq y \triangleq x.LOut \subseteq y.LOut \wedge x.LIn \subseteq y.LIn \qquad \bot \triangleq l \mapsto (\emptyset, \emptyset)$$

L'analyse des durées de vies, une analyse dataflow

Dans l'exemple, on a trouvé un point fixe en itérant F. Était-ce garanti ?

Oui ! Pourquoi ?

- P est un ensemble fini ordonné et muni d'un plus petit élément $\bot$.

$$x \sqsubseteq y \triangleq x.LOut \subseteq y.LOut \wedge x.LIn \subseteq y.LIn \qquad \bot \triangleq l \mapsto (\emptyset, \emptyset)$$

L'analyse des durées de vies, une analyse dataflow

Dans l'exemple, on a trouvé un point fixe en itérant F. Était-ce garanti ?

Oui ! Pourquoi ?

- P est un ensemble fini ordonné et muni d'un plus petit élément $\bot$.

$$x \sqsubseteq y \triangleq x.LOut \subseteq y.LOut \wedge x.LIn \subseteq y.LIn \qquad \bot \triangleq l \mapsto (\emptyset, \emptyset)$$

- Les fonctions de transfert sont monotones.

$$\forall x \sqsubseteq y, F_l(x) \sqsubseteq F_l(y)$$

L'analyse des durées de vies, une analyse dataflow

Dans l'exemple, on a trouvé un point fixe en itérant F. Était-ce garanti ?

Oui ! Pourquoi ?

- P est un ensemble fini ordonné et muni d'un plus petit élément $\bot$.

$$x \sqsubseteq y \triangleq x.LOut \subseteq y.LOut \wedge x.LIn \subseteq y.LIn \qquad \bot \triangleq l \mapsto (\emptyset, \emptyset)$$

- Les fonctions de transfert sont monotones.

$$\forall x \sqsubseteq y, F_l(x) \sqsubseteq F_l(y)$$

- Par conséquent, il existe un entier n_0 tel que pour tout $n \geq n_0$ on ait :

$$F^n(\bot) = F^{n+1}(\bot).$$

On peut aussi montrer que $F^{n_0}(\bot)$ est le plus petit point fixe de F.

L'analyse des durées de vies, une analyse dataflow

Dans l'exemple, on a trouvé un point fixe en itérant F. Était-ce garanti ?

Oui ! Pourquoi ?

- P est un ensemble fini ordonné et muni d'un plus petit élément $\bot$.

$$x \sqsubseteq y \triangleq x.LOut \subseteq y.LOut \wedge x.LIn \subseteq y.LIn \qquad \bot \triangleq l \mapsto (\emptyset, \emptyset)$$

- Les fonctions de transfert sont monotones.

$$\forall x \sqsubseteq y, F_l(x) \sqsubseteq F_l(y)$$

- Par conséquent, il existe un entier n_0 tel que pour tout $n \geq n_0$ on ait :

$$F^n(\bot) = F^{n+1}(\bot).$$

  On peut aussi montrer que $F^{n_0}(\bot)$ est le plus petit point fixe de F.

Les *analyses dataflow* se placent dans des variations de ce cadre.

Le cadre classique de l'analyse dataflow

Analyse dataflow

Analyse qui associe une propriété statique approximant le comportement dynamique du programme en chacun de ses points, calculée itérativement.

Elle est donnée par :

Le cadre classique de l'analyse dataflow

Analyse dataflow

Analyse qui associe une propriété statique approximant le comportement dynamique du programme en chacun de ses points, calculée itérativement.

Elle est donnée par :

- Un ensemble de *propriétés*, demi-treillis de hauteur finie.

$$(P, \sqsubseteq, \bot, \sqcup)$$

Le cadre classique de l'analyse dataflow

Analyse dataflow

Analyse qui associe une propriété statique approximant le comportement dynamique du programme en chacun de ses points, calculée itérativement.

Elle est donnée par :

- Un ensemble de *propriétés*, demi-treillis de hauteur finie.

$$(P, \sqsubseteq, \bot, \sqcup)$$

- Une *fonction de transfert* monotone par point de programme l.

$$F_l : (Lab \to P) \to P$$

Le cadre classique de l'analyse dataflow

Analyse dataflow

Analyse qui associe une propriété statique approximant le comportement dynamique du programme en chacun de ses points, calculée itérativement.

Elle est donnée par :

- Un ensemble de *propriétés*, demi-treillis de hauteur finie.

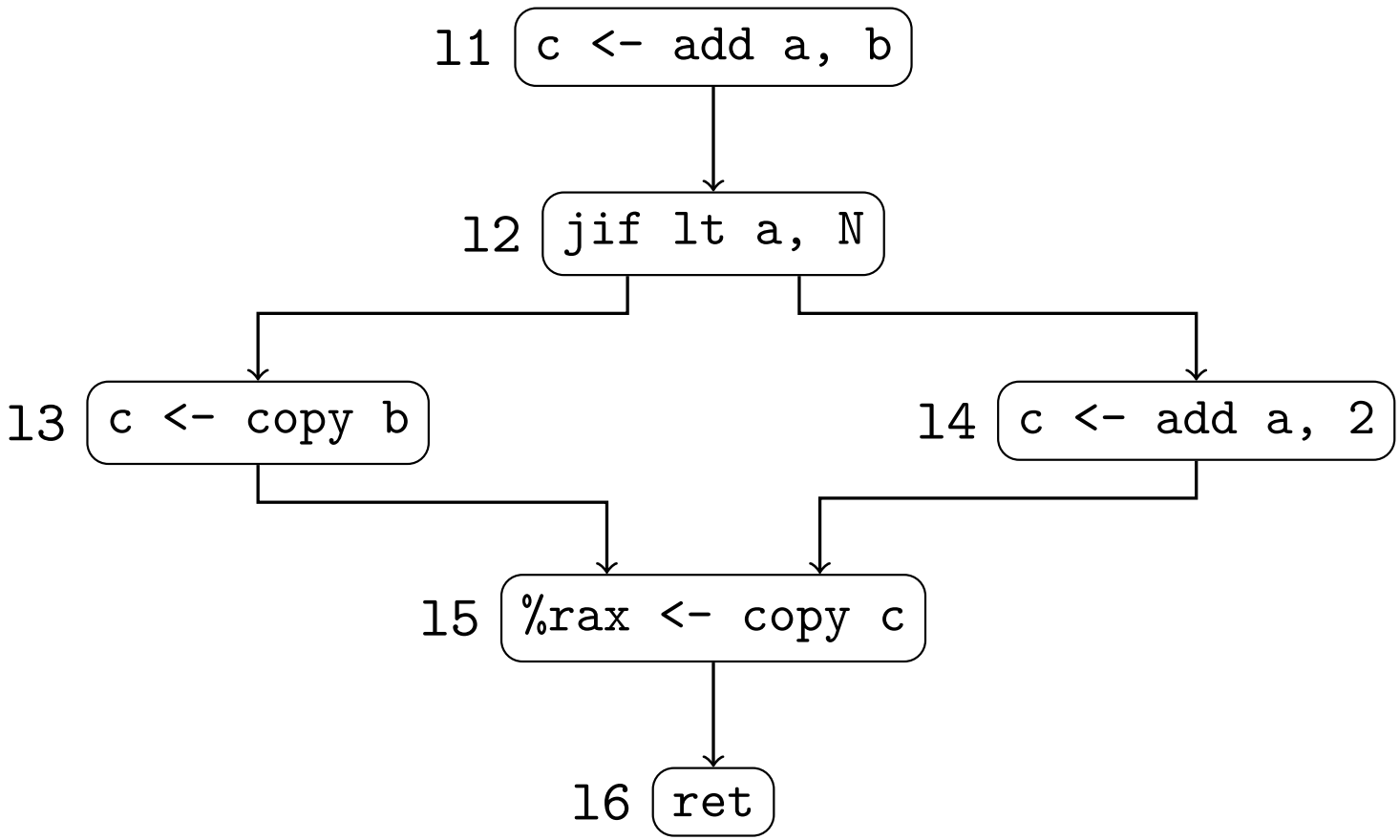
$$(P, \sqsubseteq, \bot, \sqcup)$$

- Une *fonction de transfert* monotone par point de programme l.

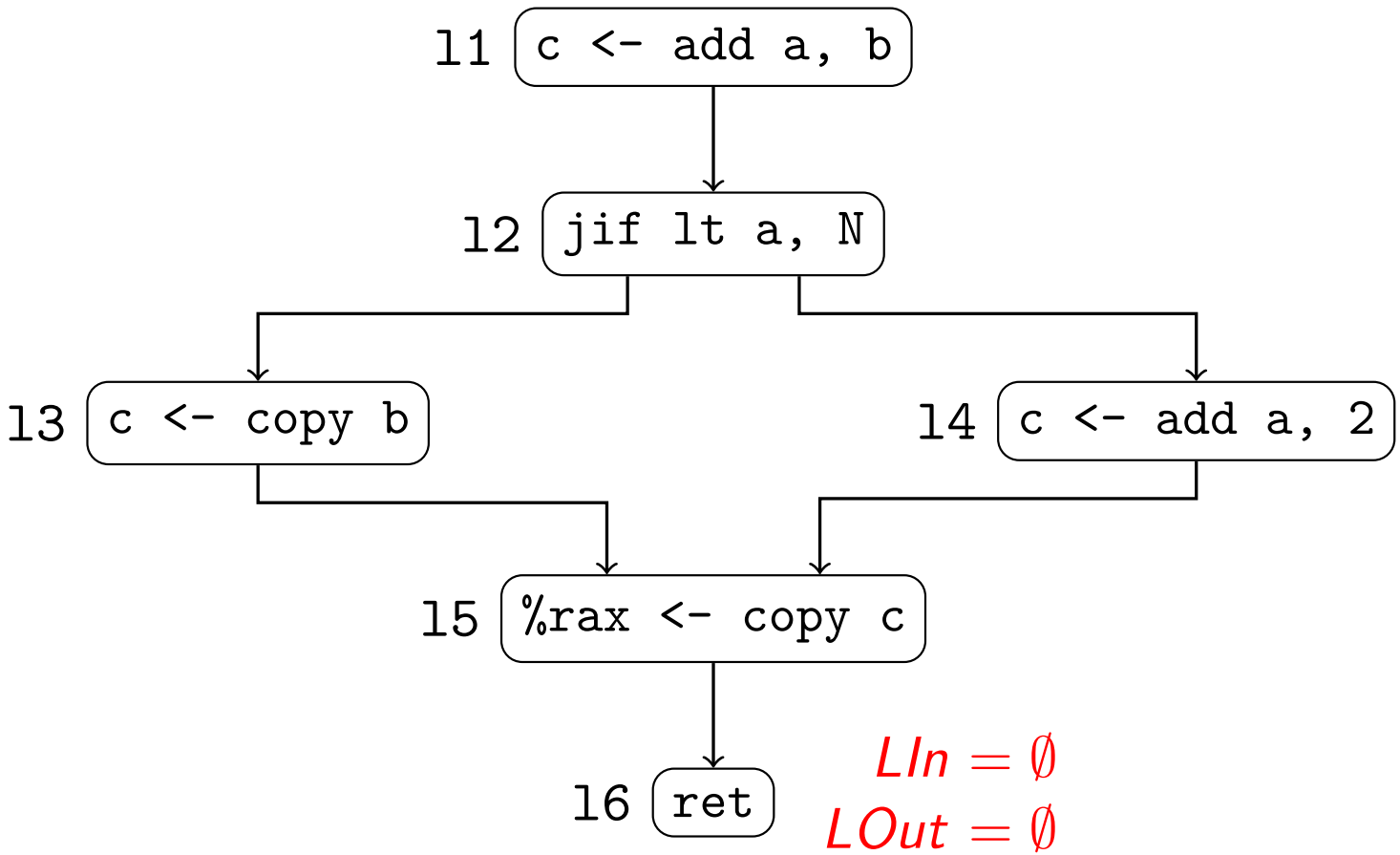
$$F_l : (Lab \to P) \to P$$

- Une *stratégie d'itération* pour accélerer le calcul du point fixe.
  (Par exemple, dans l'analyse des durées de vie, calculer en arrière.)

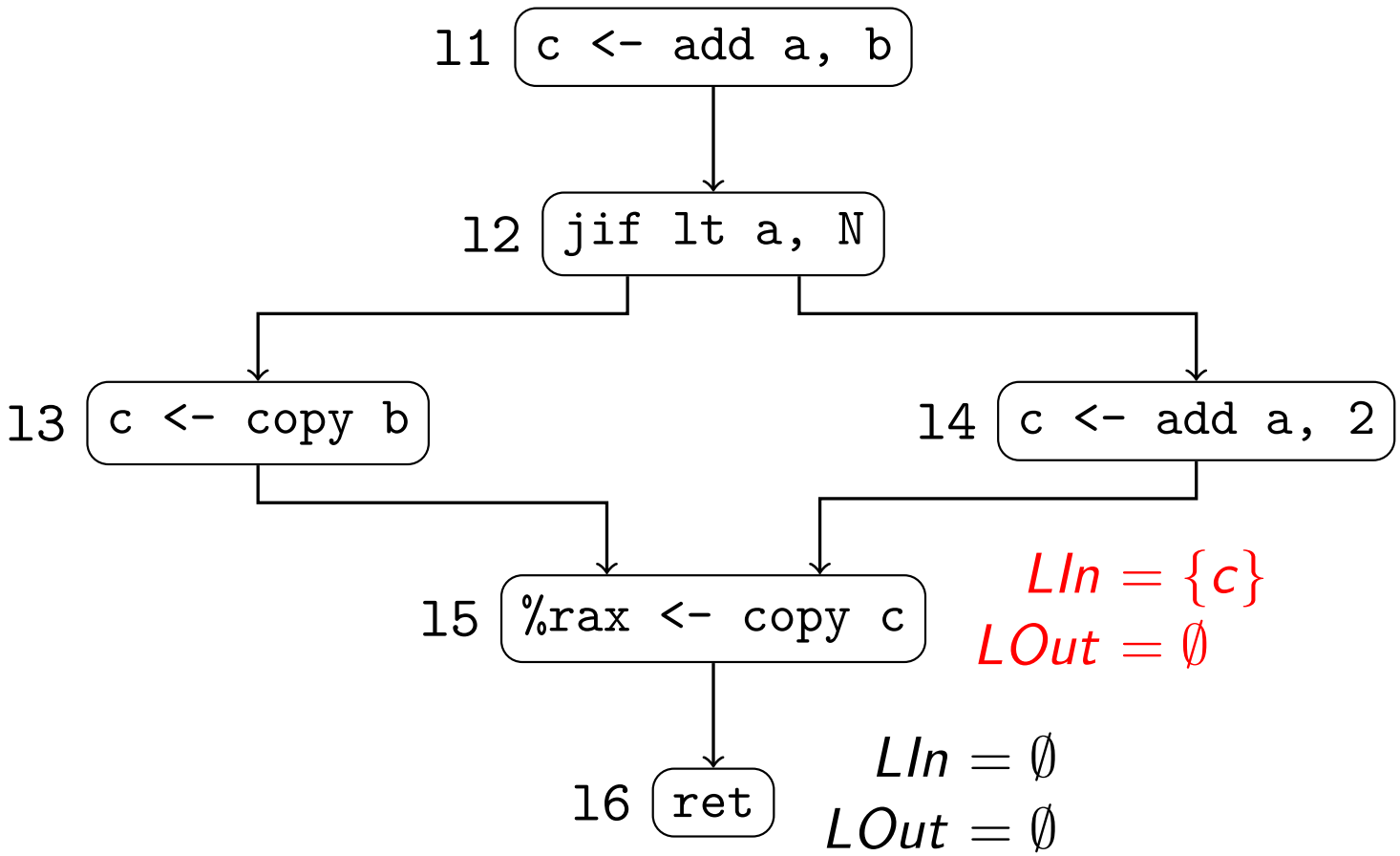
Analyses dataflow au service d'optimisations : DCE



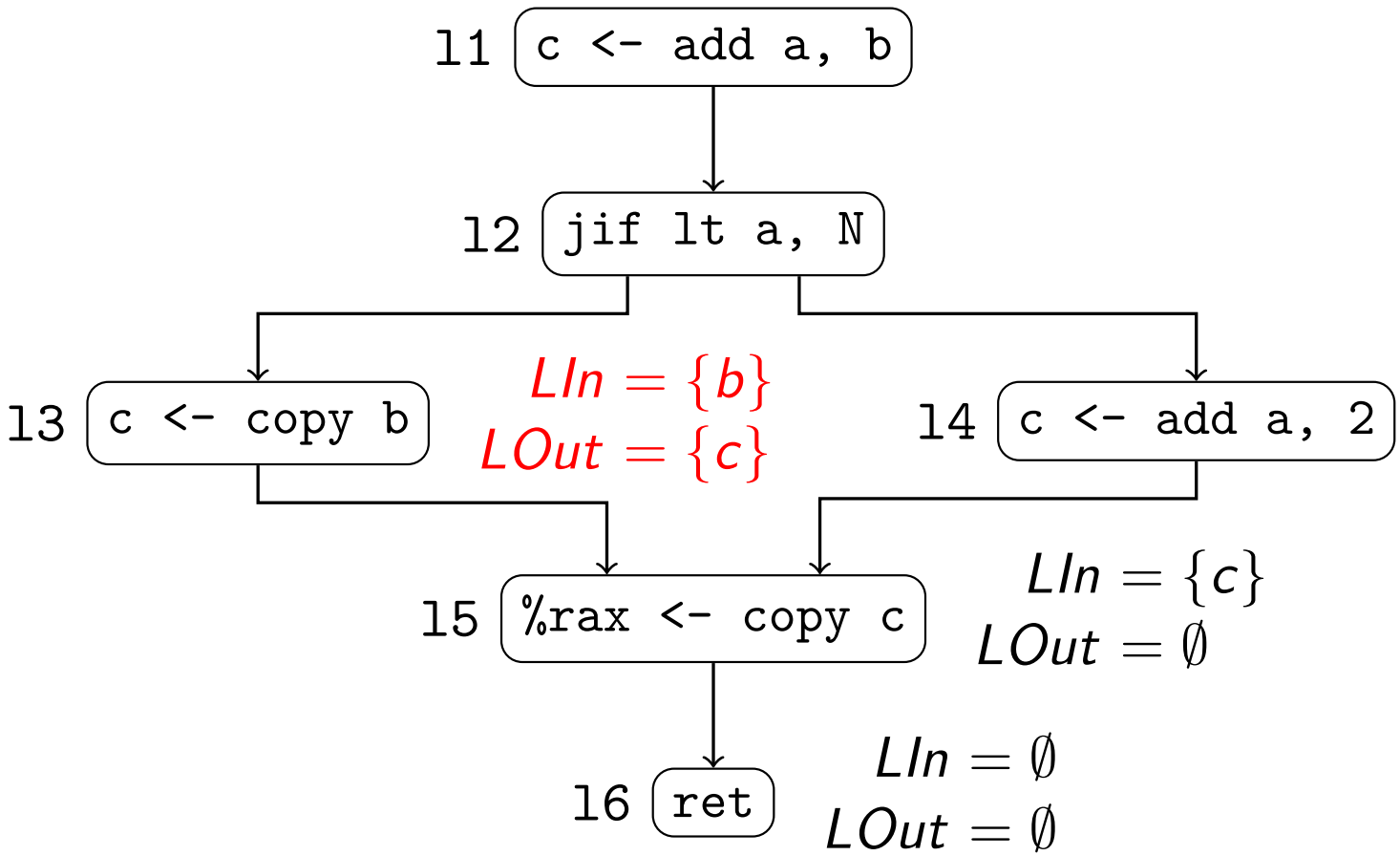
Analyses dataflow au service d'optimisations : DCE



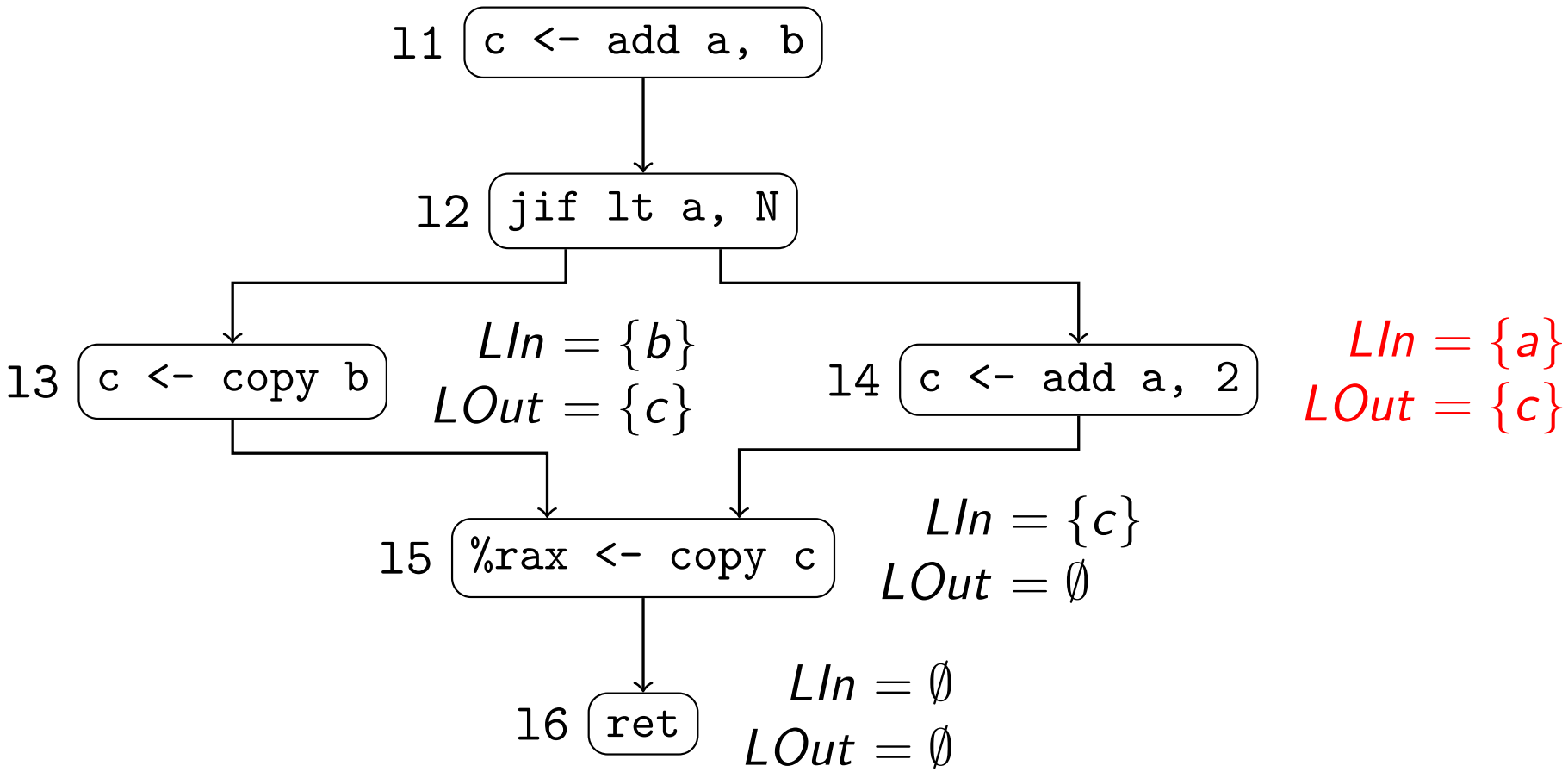
Analyses dataflow au service d'optimisations : DCE



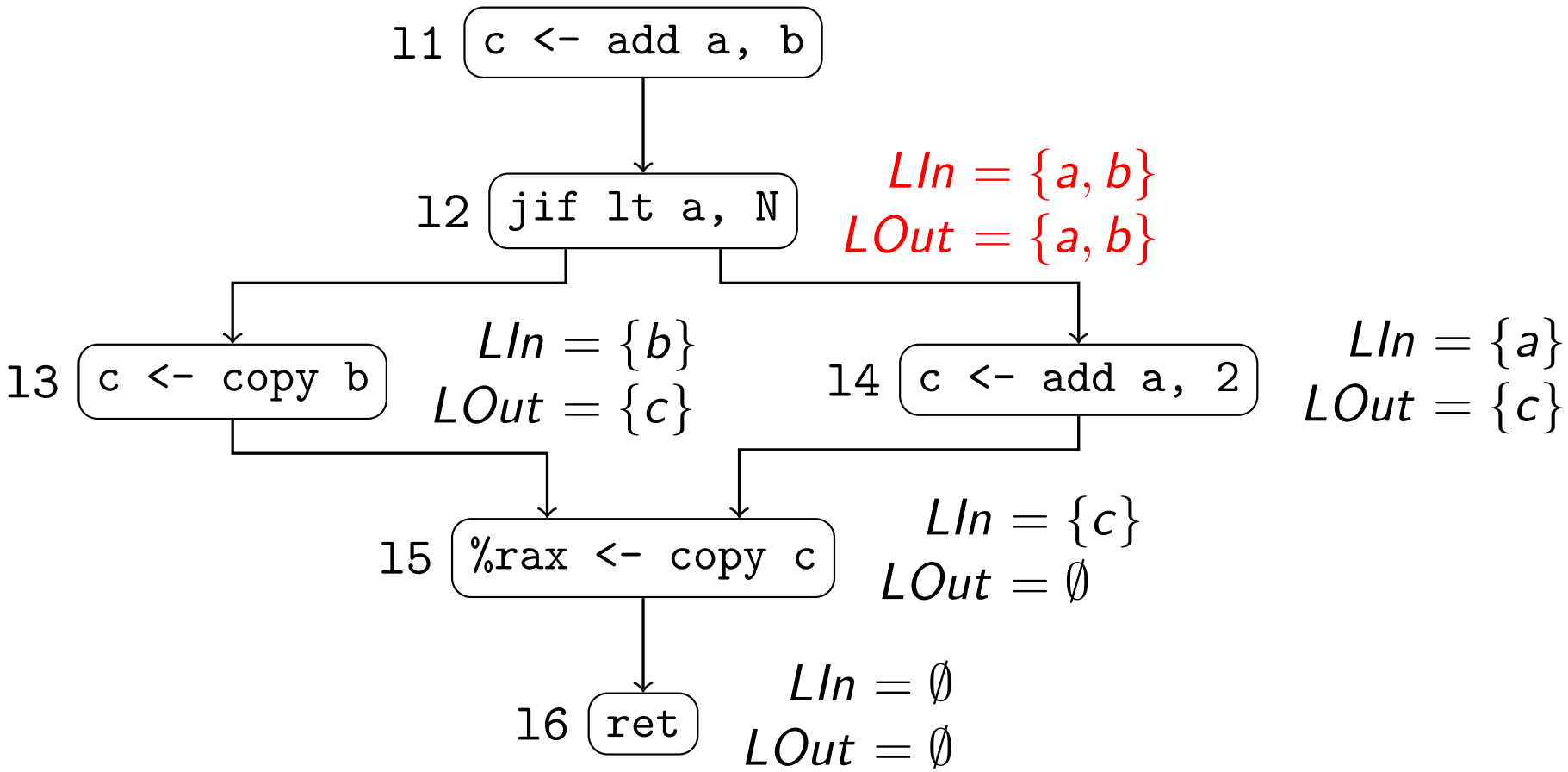
Analyses dataflow au service d'optimisations : DCE



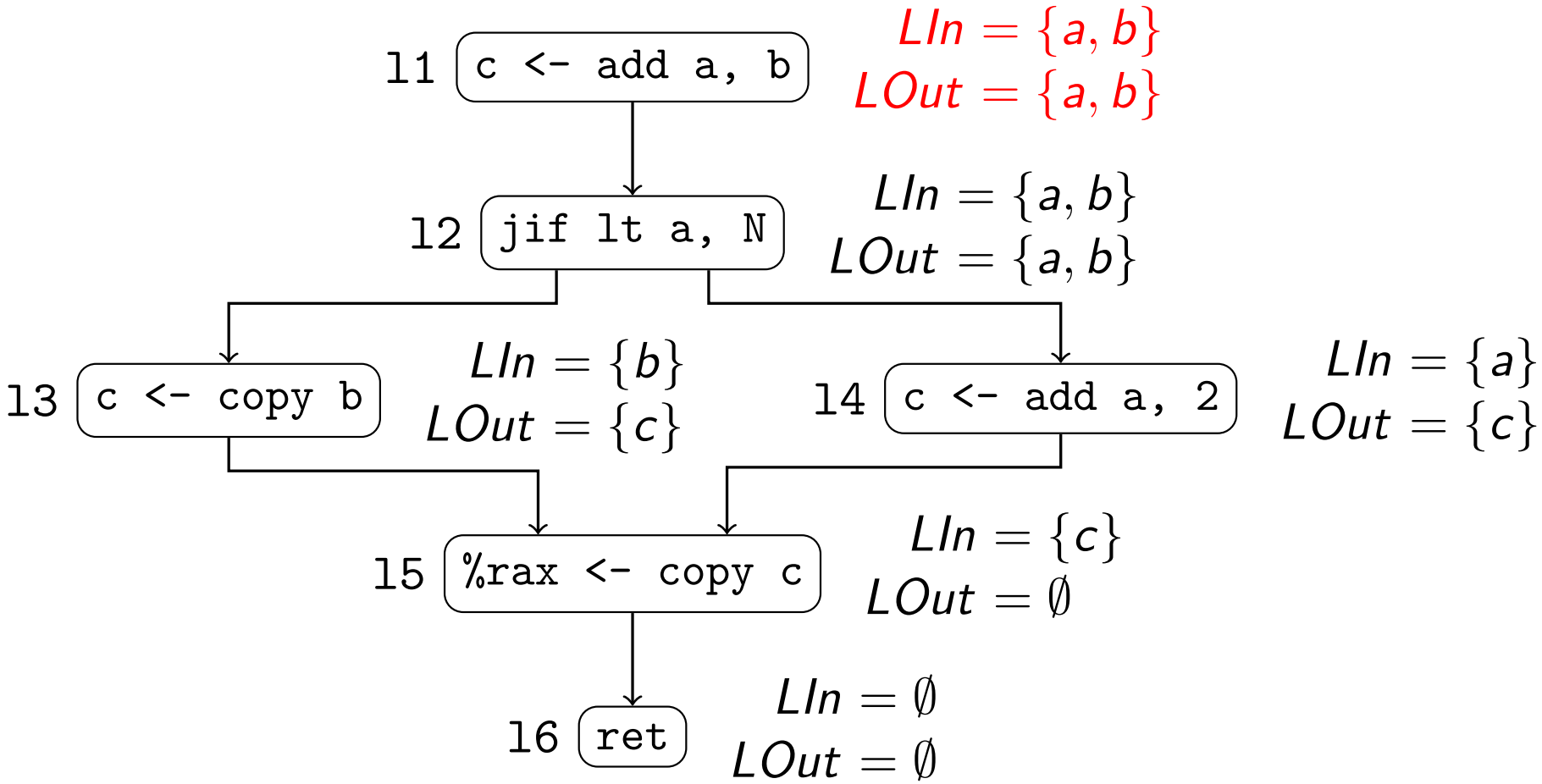
Analyses dataflow au service d'optimisations : DCE



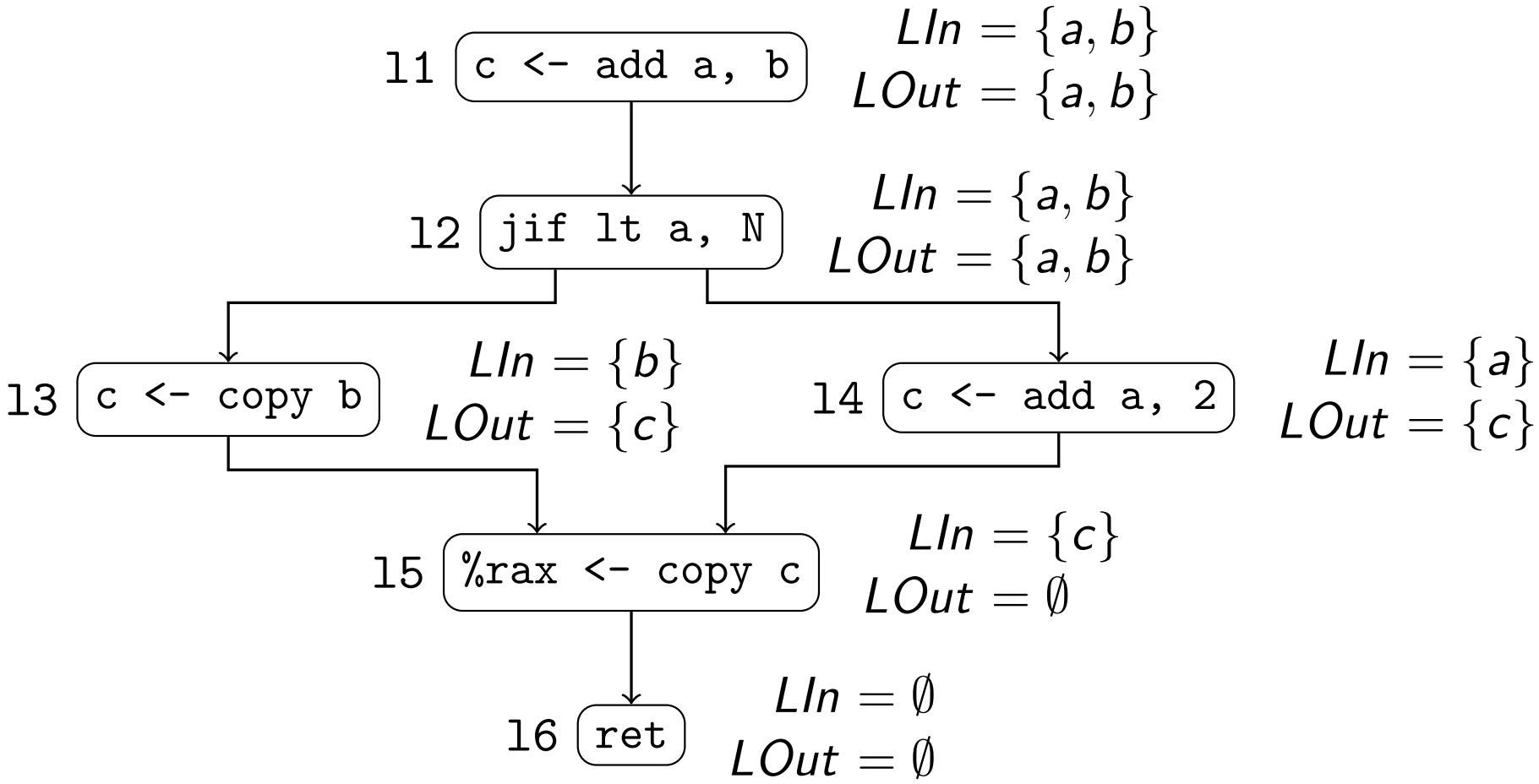
Analyses dataflow au service d'optimisations : DCE



Analyses dataflow au service d'optimisations : DCE

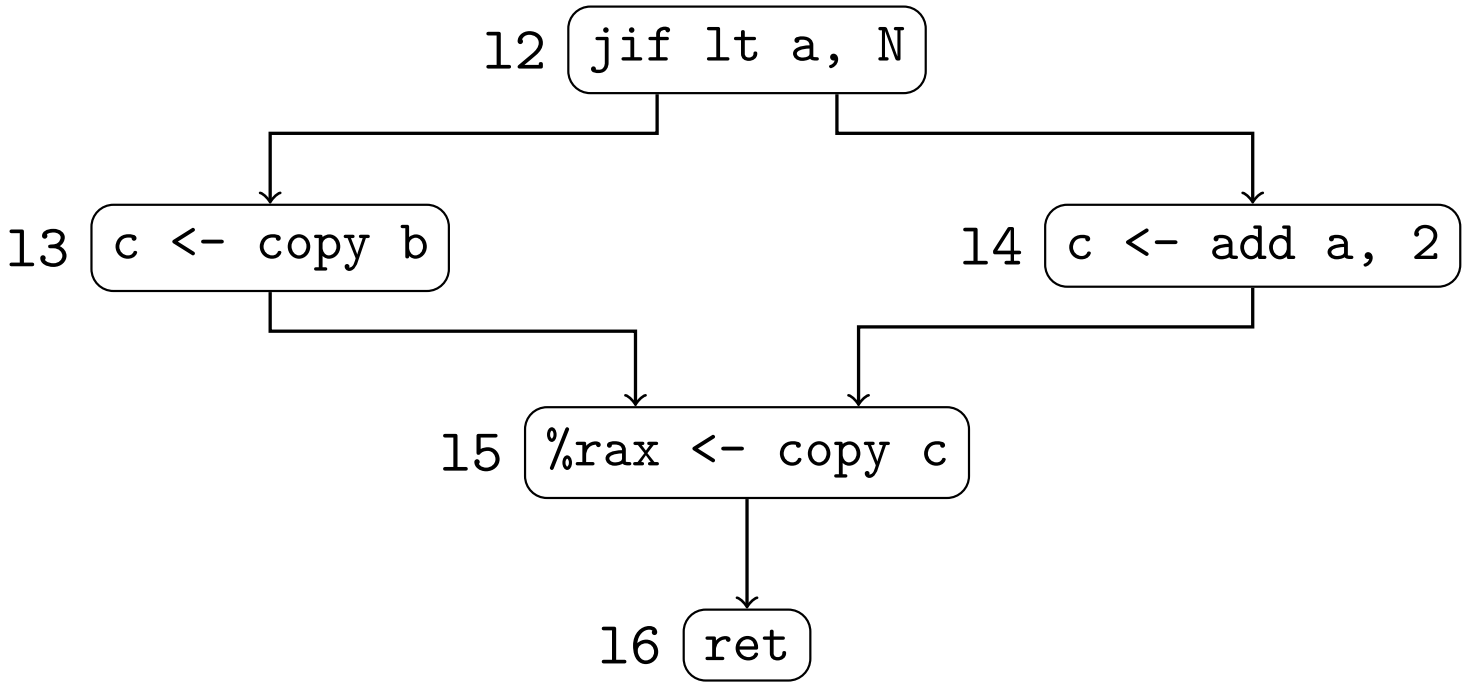


Analyses dataflow au service d'optimisations : DCE



Dead-Code Elimination : si $Def(l) \cap LOut(l) = \emptyset$, l peut être supprimé.

Analyses dataflow au service d'optimisations : DCE



Dead-Code Elimination : si $Def(l) \cap LOut(l) = \emptyset$, l peut être supprimé.

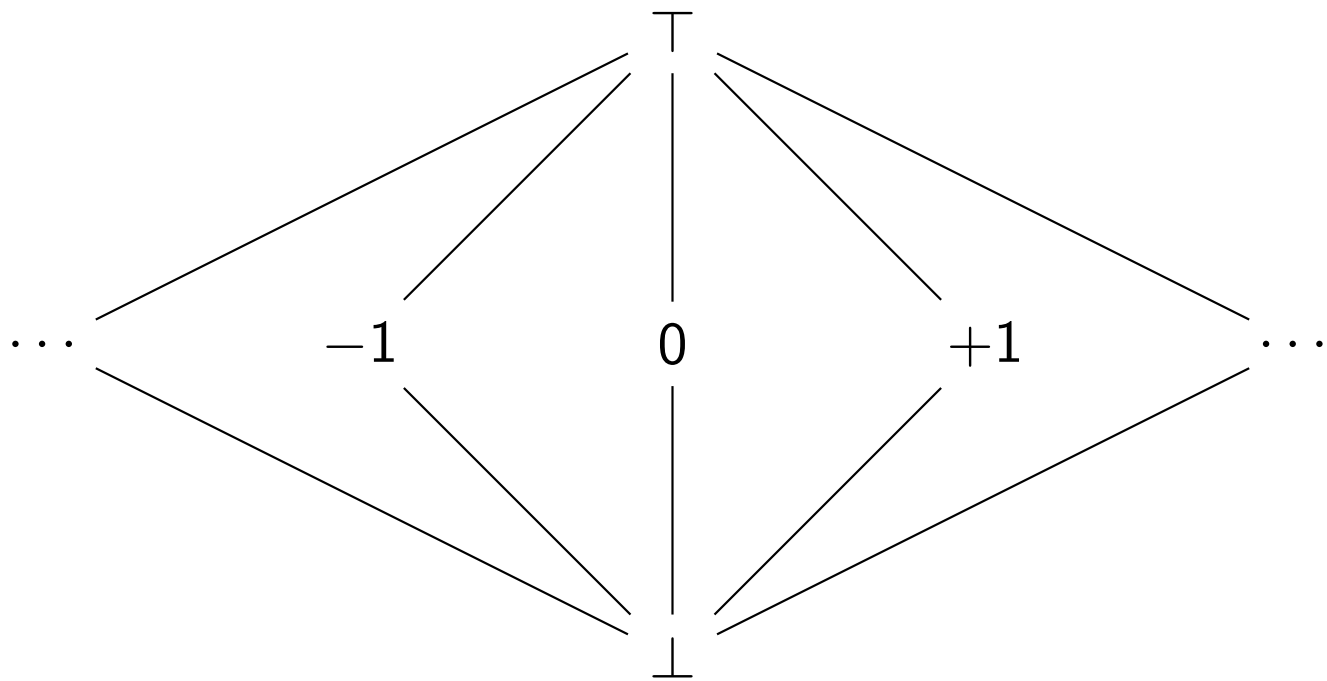
Analyses dataflow au service d'optimisations : CF (1/2)

La *propagation de constante* (*Constant Folding*) simplifie un programme en remplaçant les expressions constantes par leurs valeurs.

Analyses dataflow au service d'optimisations : CF (1/2)

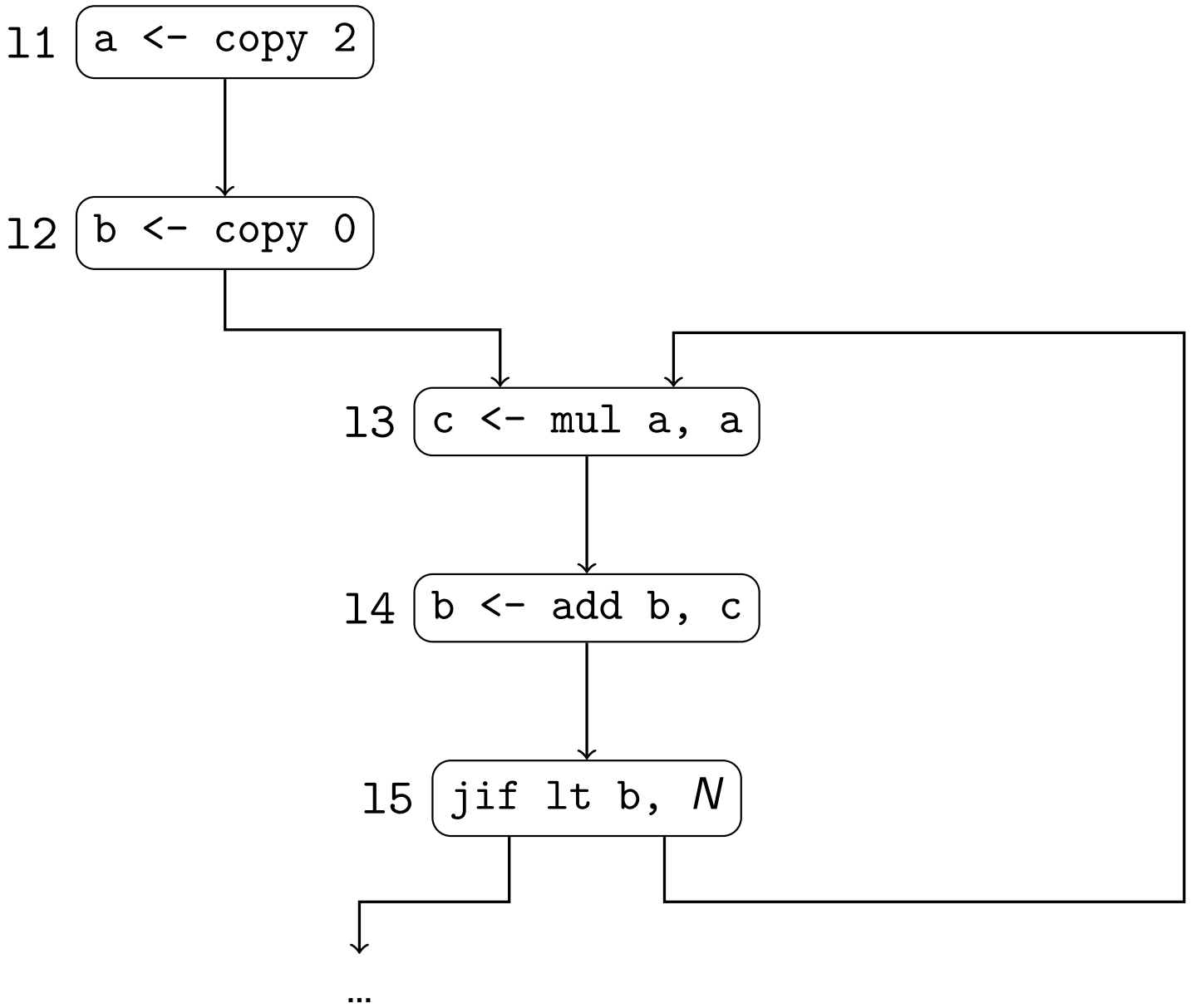
La *propagation de constante* (*Constant Folding*) simplifie un programme en remplaçant les expressions constantes par leurs valeurs.

On calcule ces valeurs par analyse dataflow en avant. Ici, nos propriétés associent à chaque variable un élément du *treillis des constantes* :

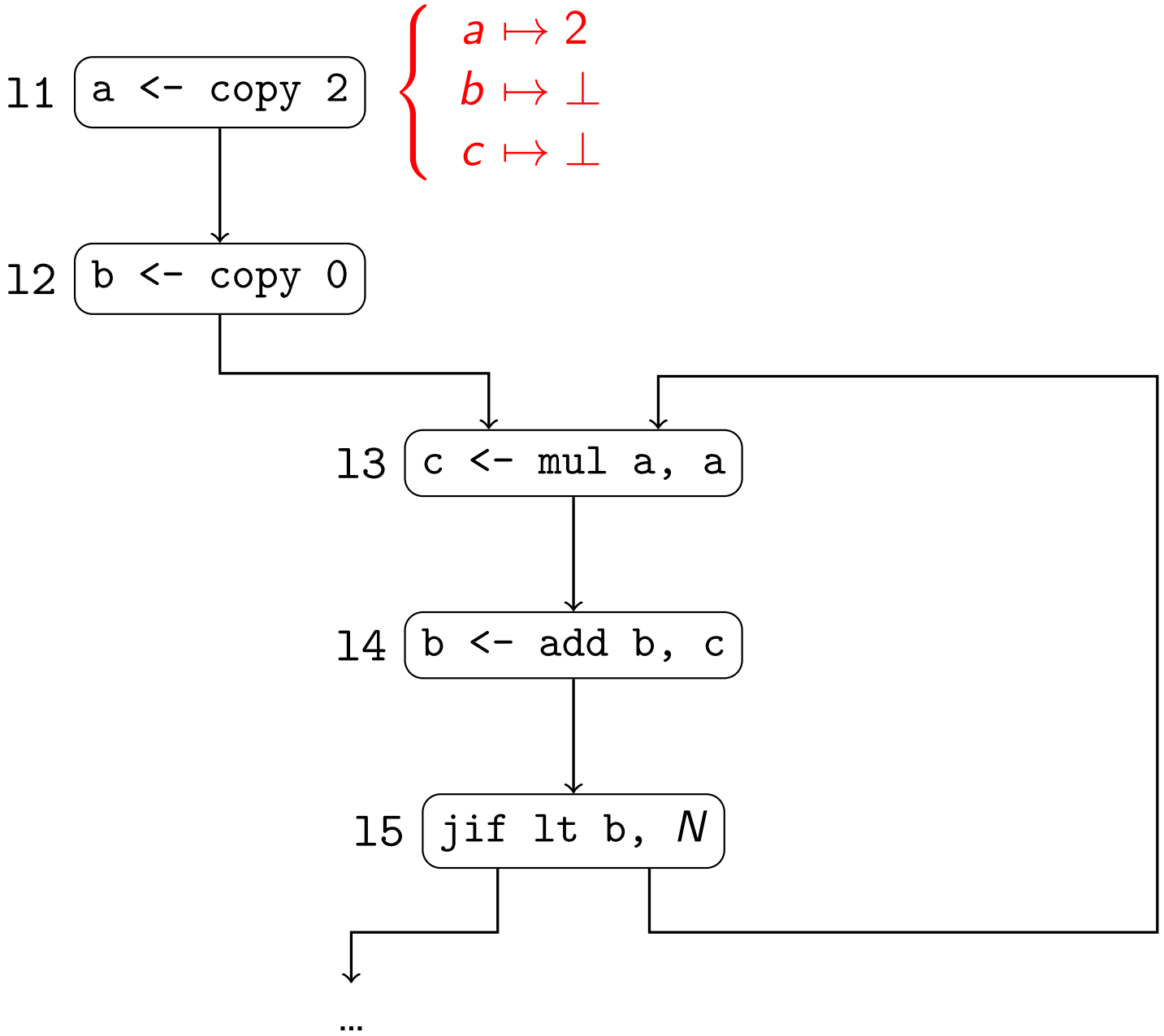


Intuitivement, $\bot$ signifie "non-initialisé" et $\top$ "non-constant".

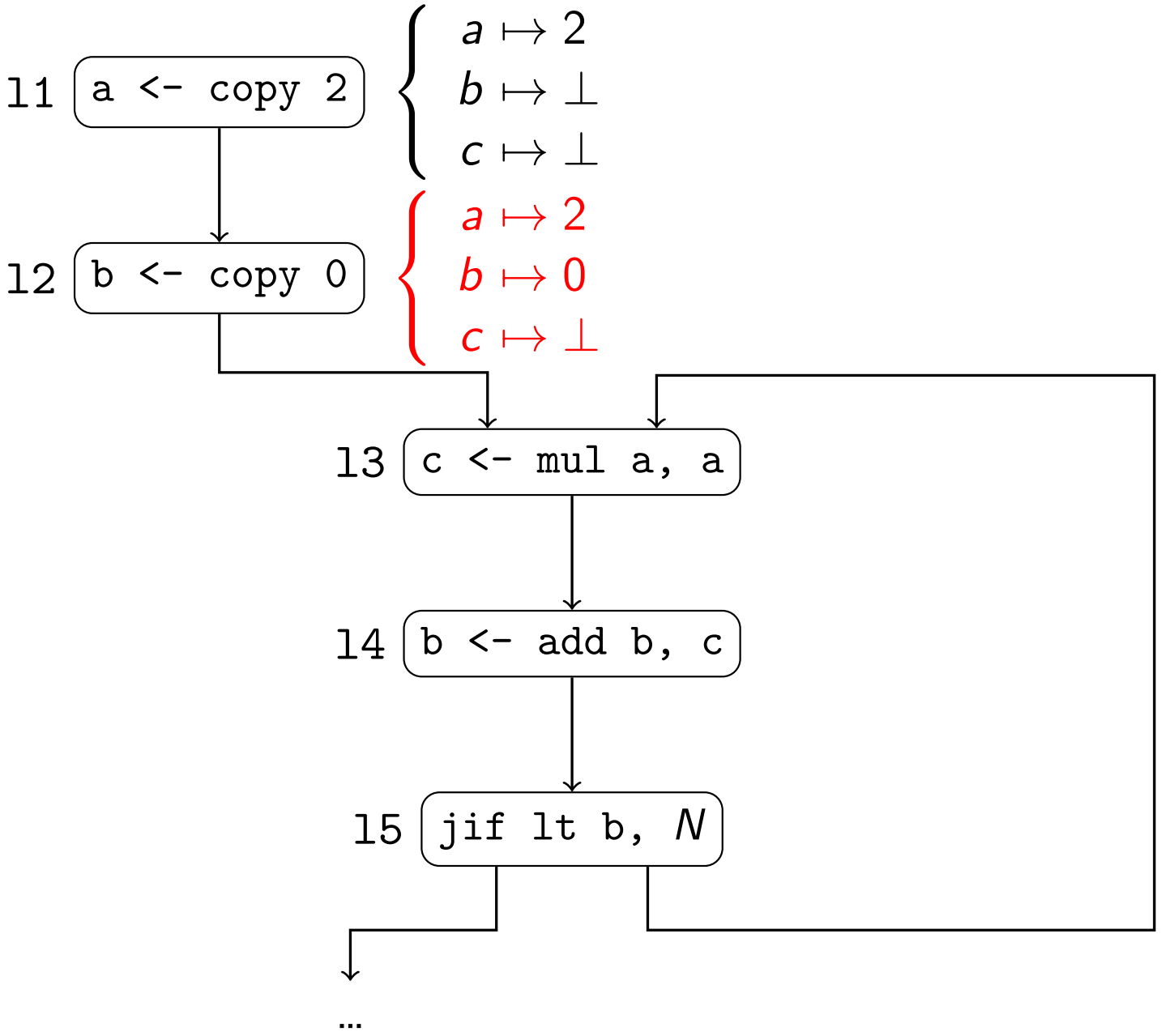
Analyses dataflow au service d'optimisations : CF (2/2)



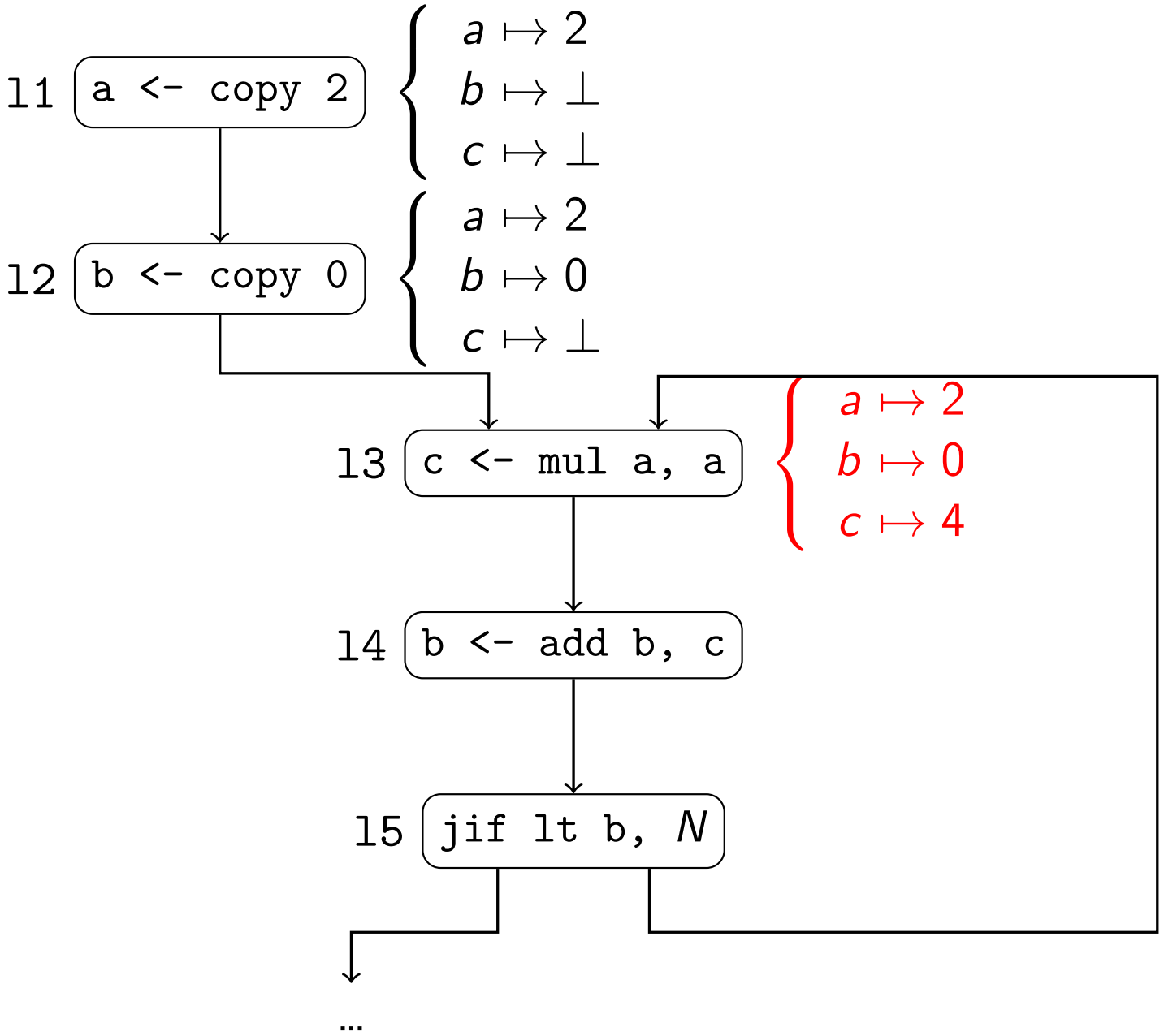
Analyses dataflow au service d'optimisations : CF (2/2)



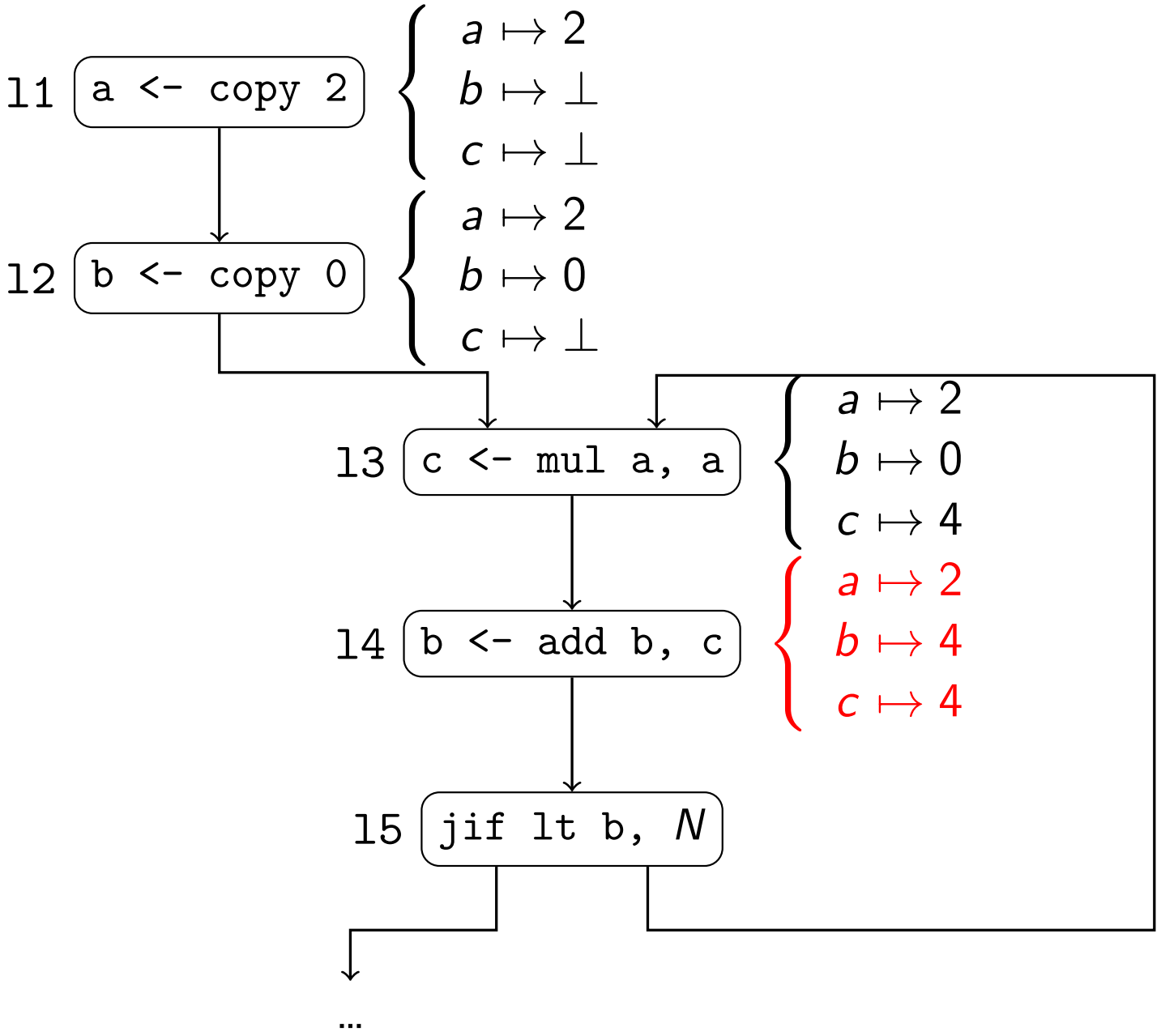
Analyses dataflow au service d'optimisations : CF (2/2)



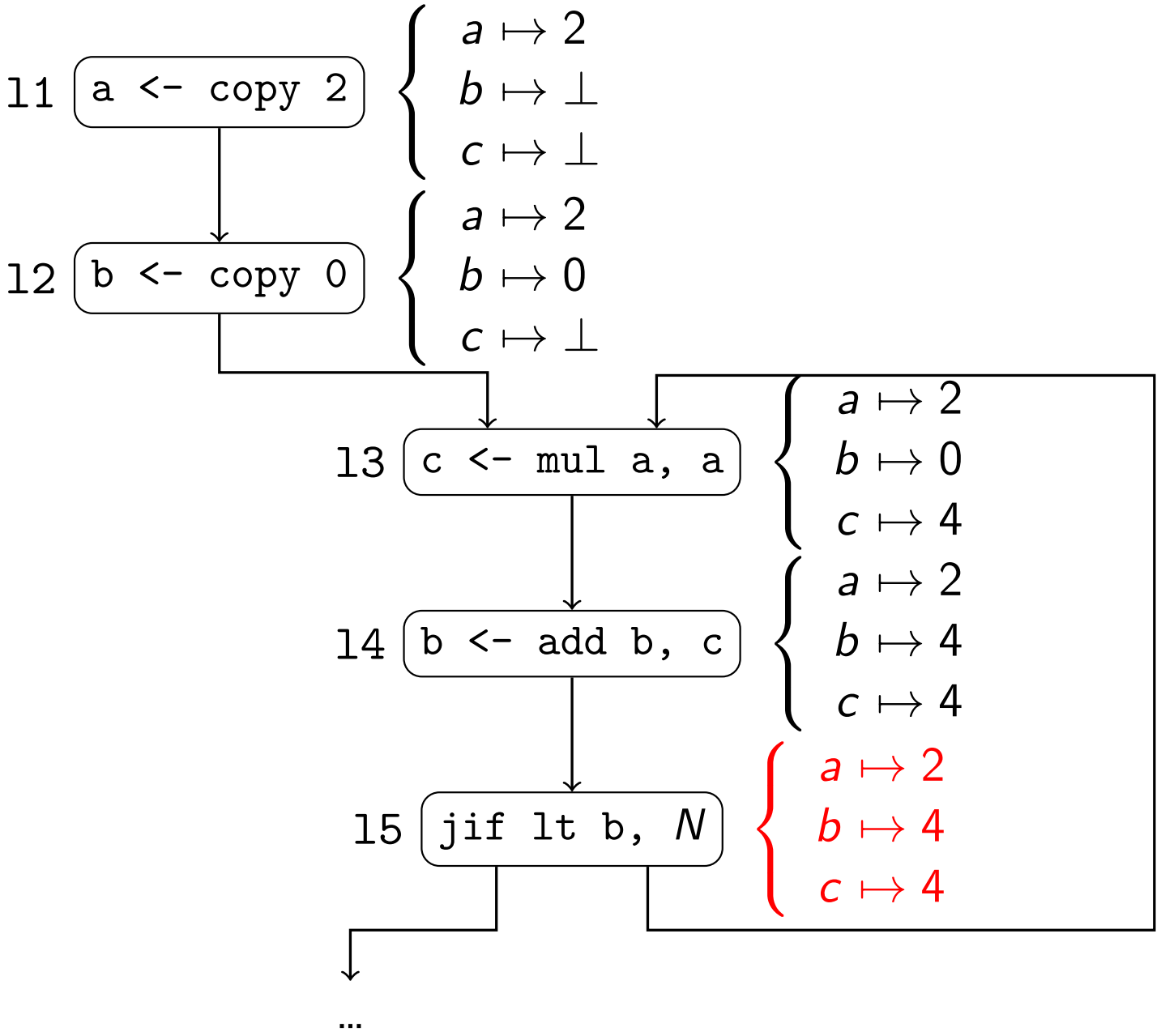
Analyses dataflow au service d'optimisations : CF (2/2)



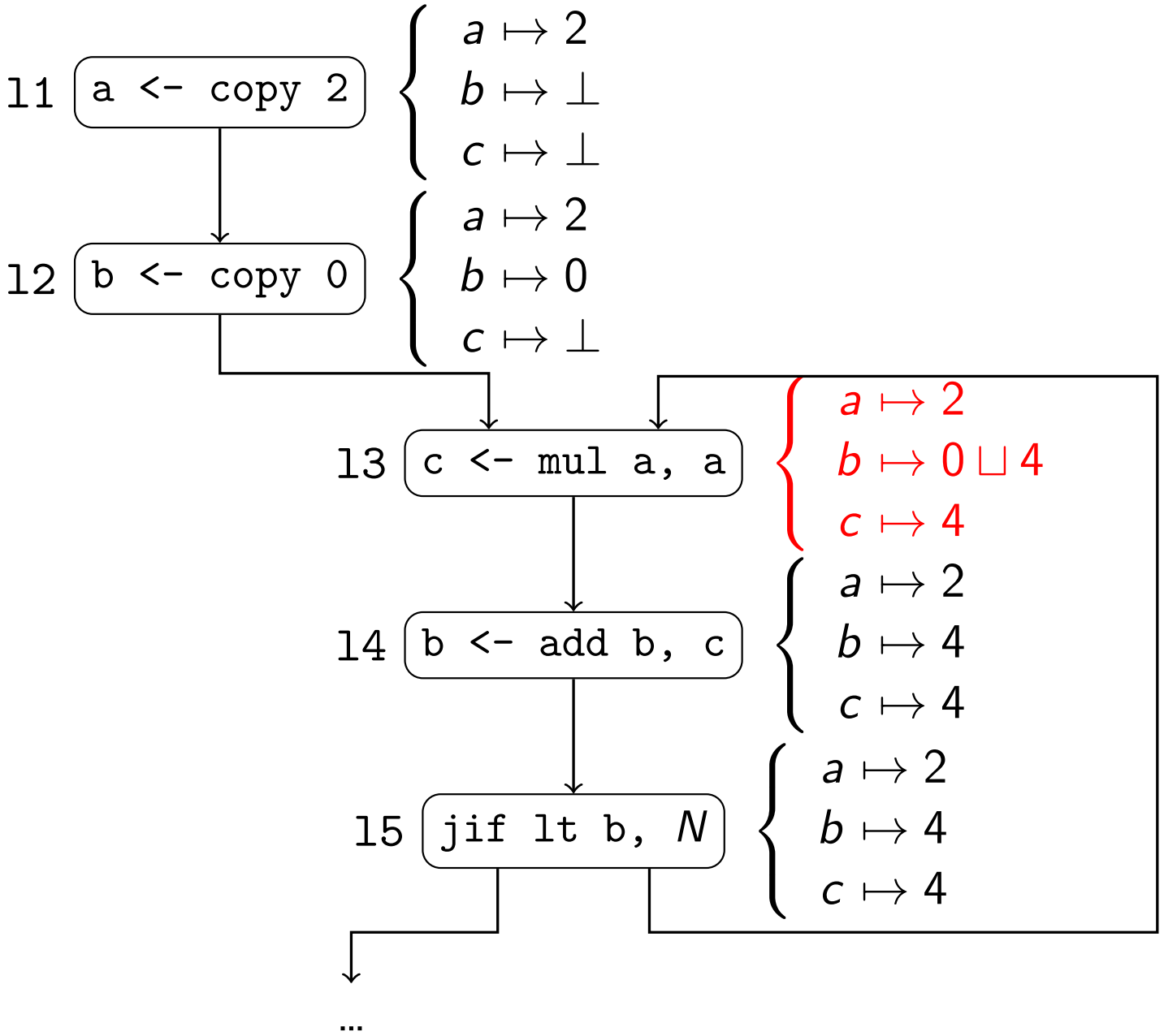
Analyses dataflow au service d'optimisations : CF (2/2)



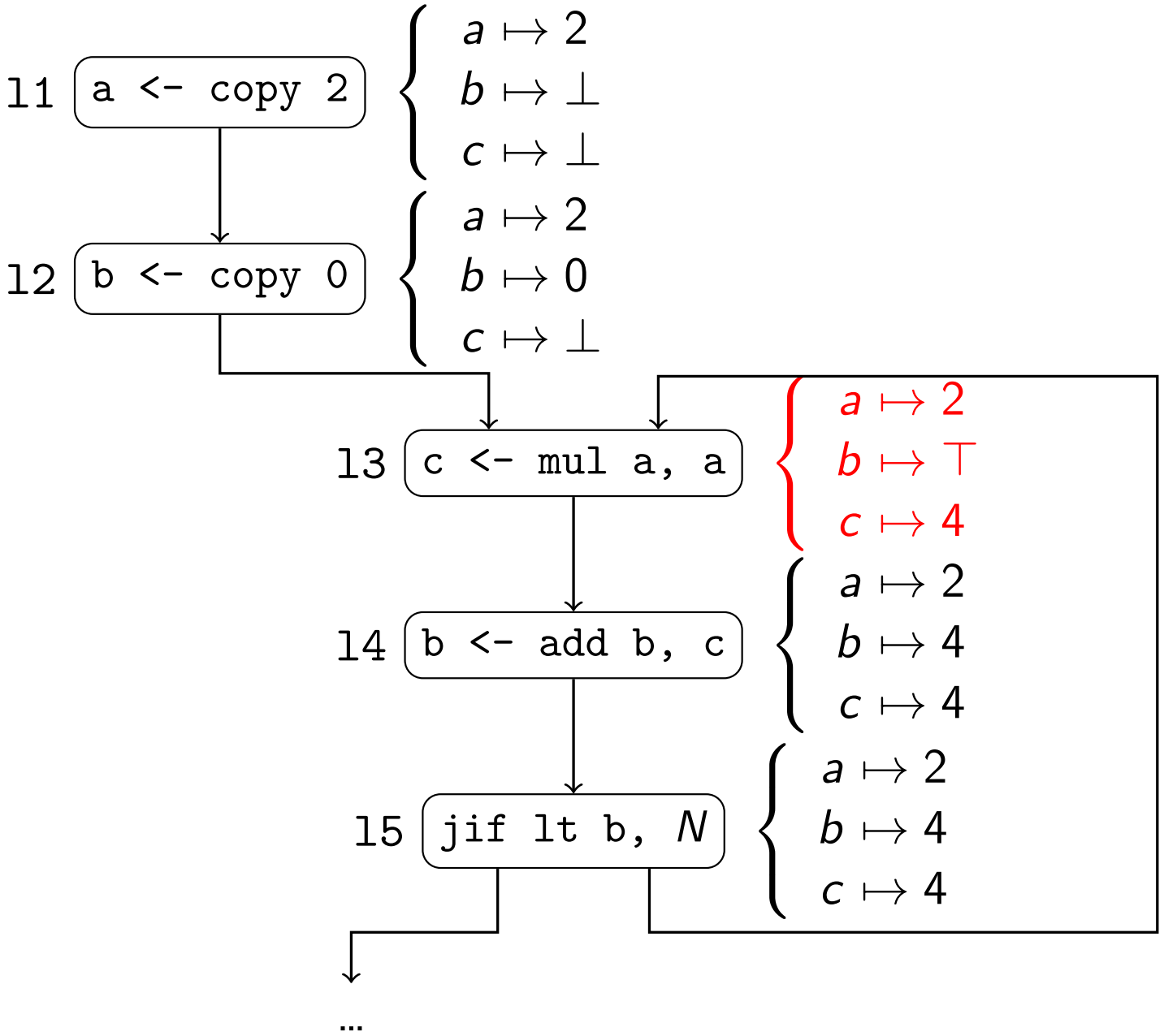
Analyses dataflow au service d'optimisations : CF (2/2)



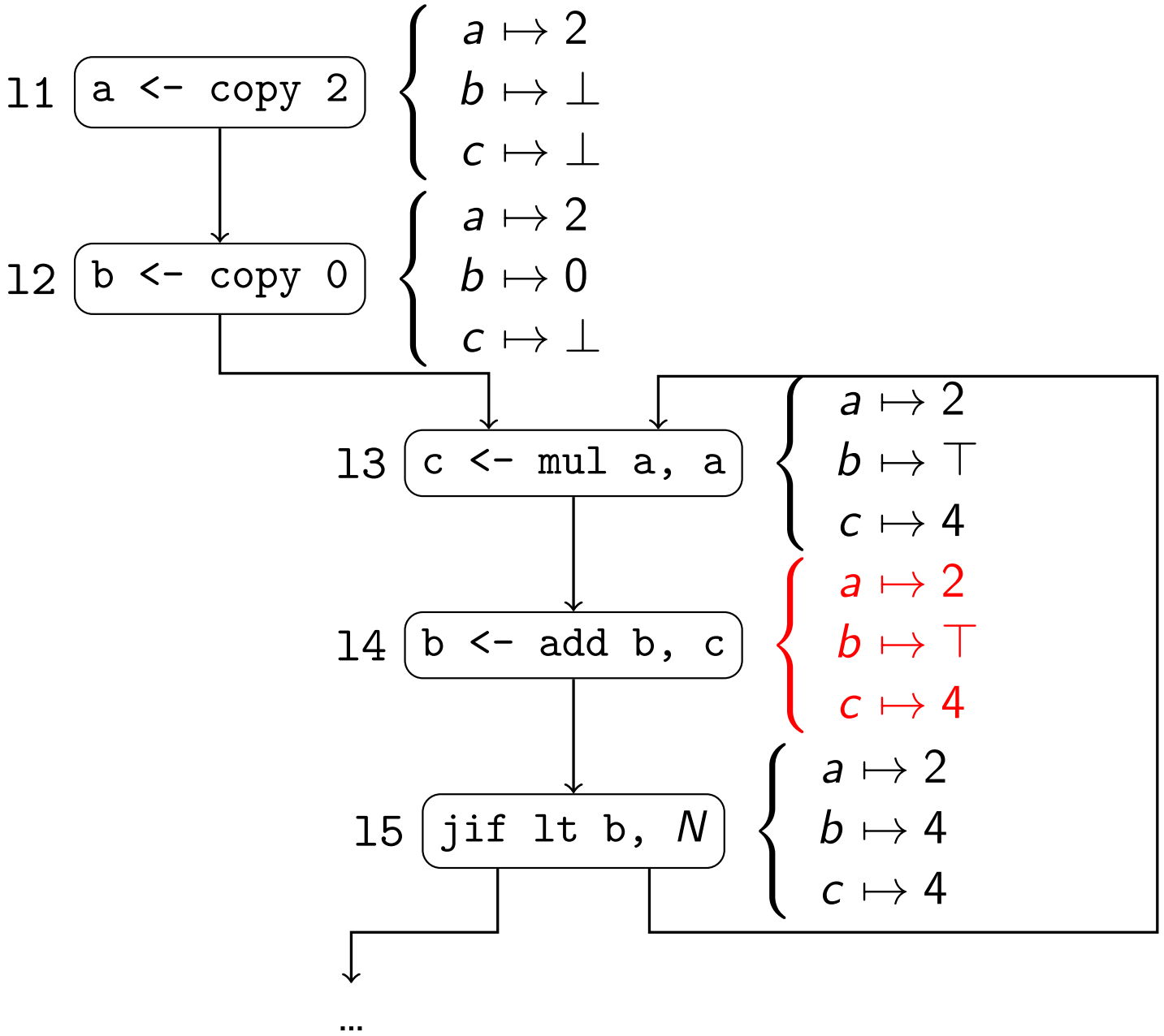
Analyses dataflow au service d'optimisations : CF (2/2)



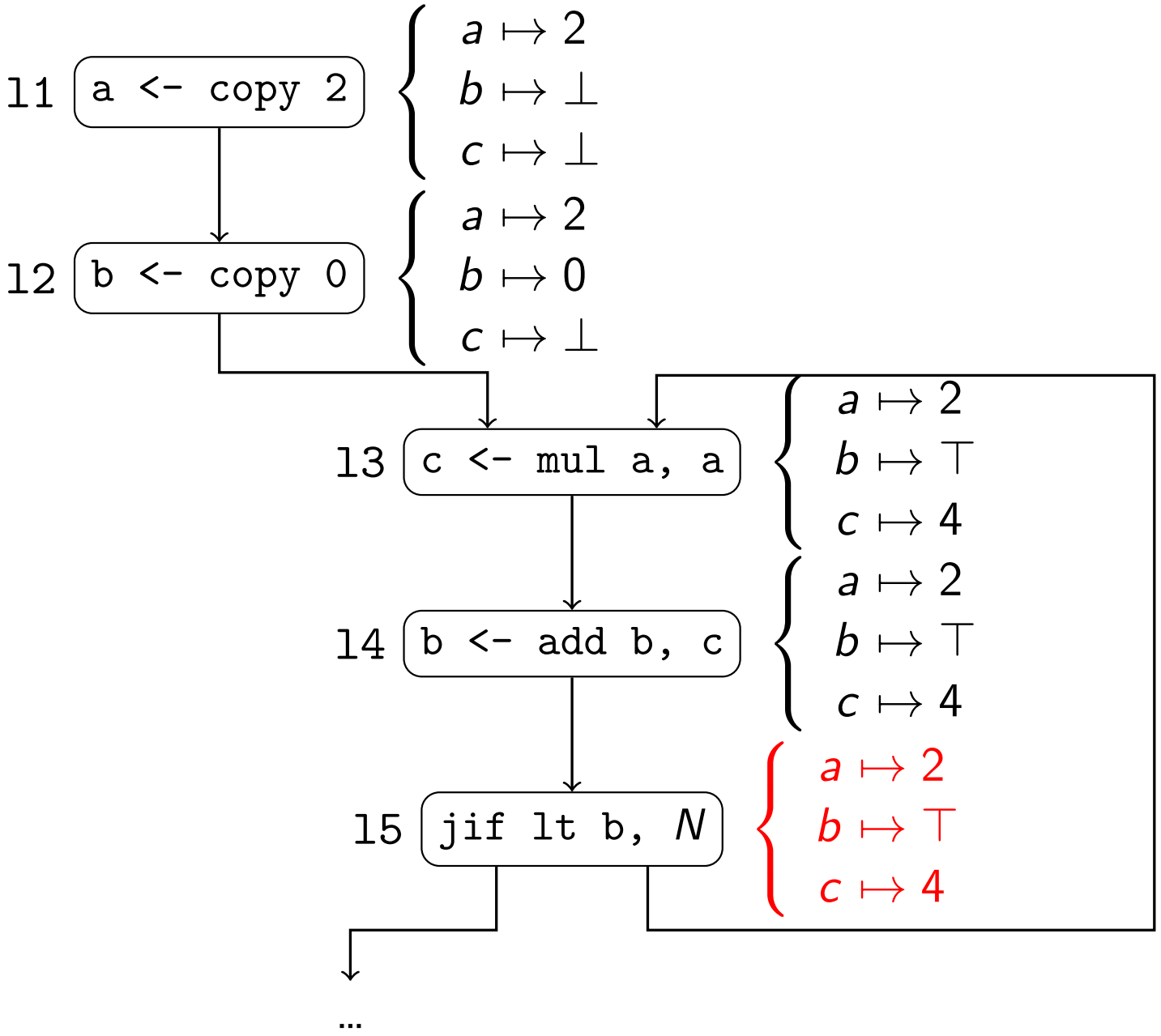
Analyses dataflow au service d'optimisations : CF (2/2)



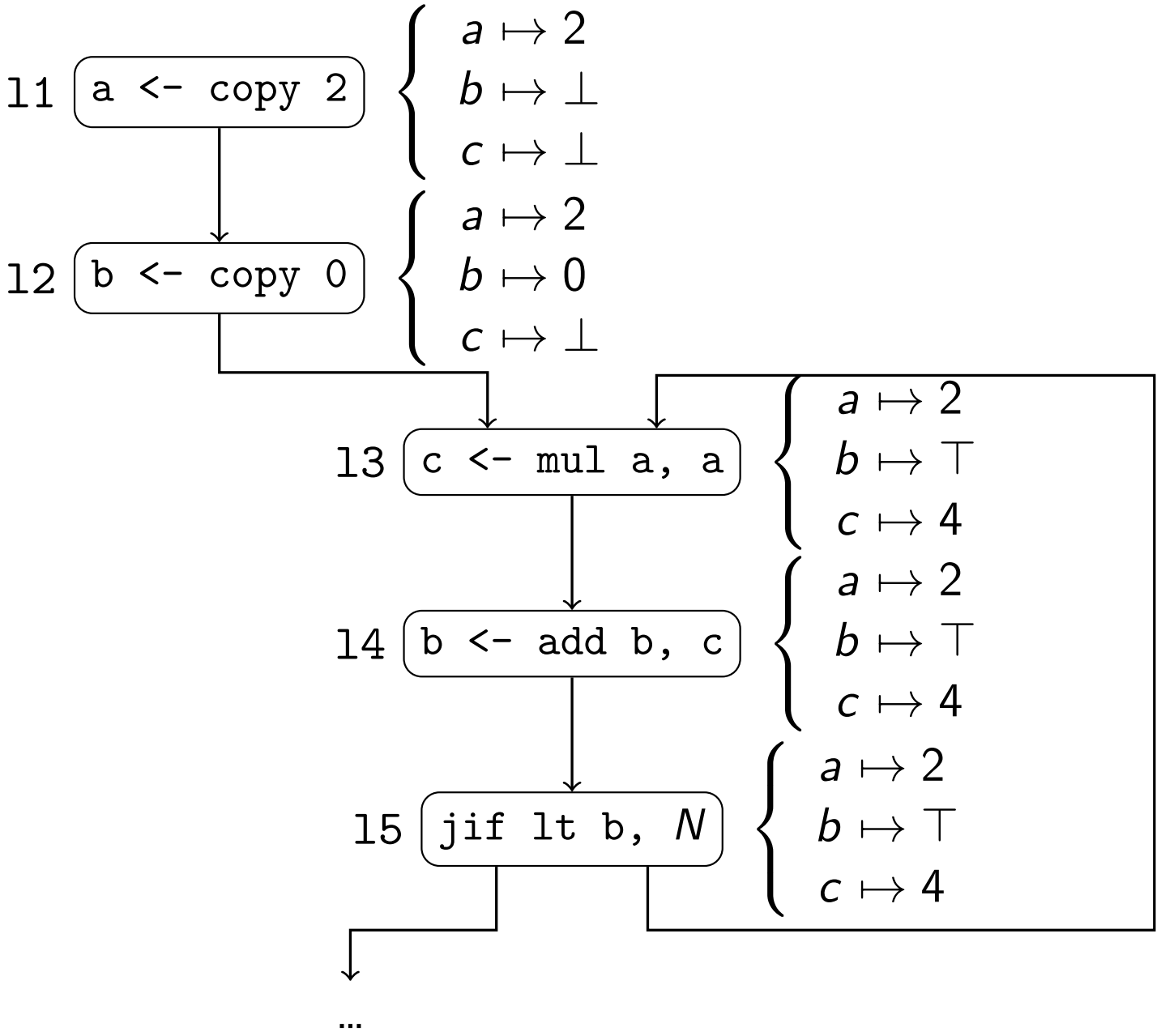
Analyses dataflow au service d'optimisations : CF (2/2)



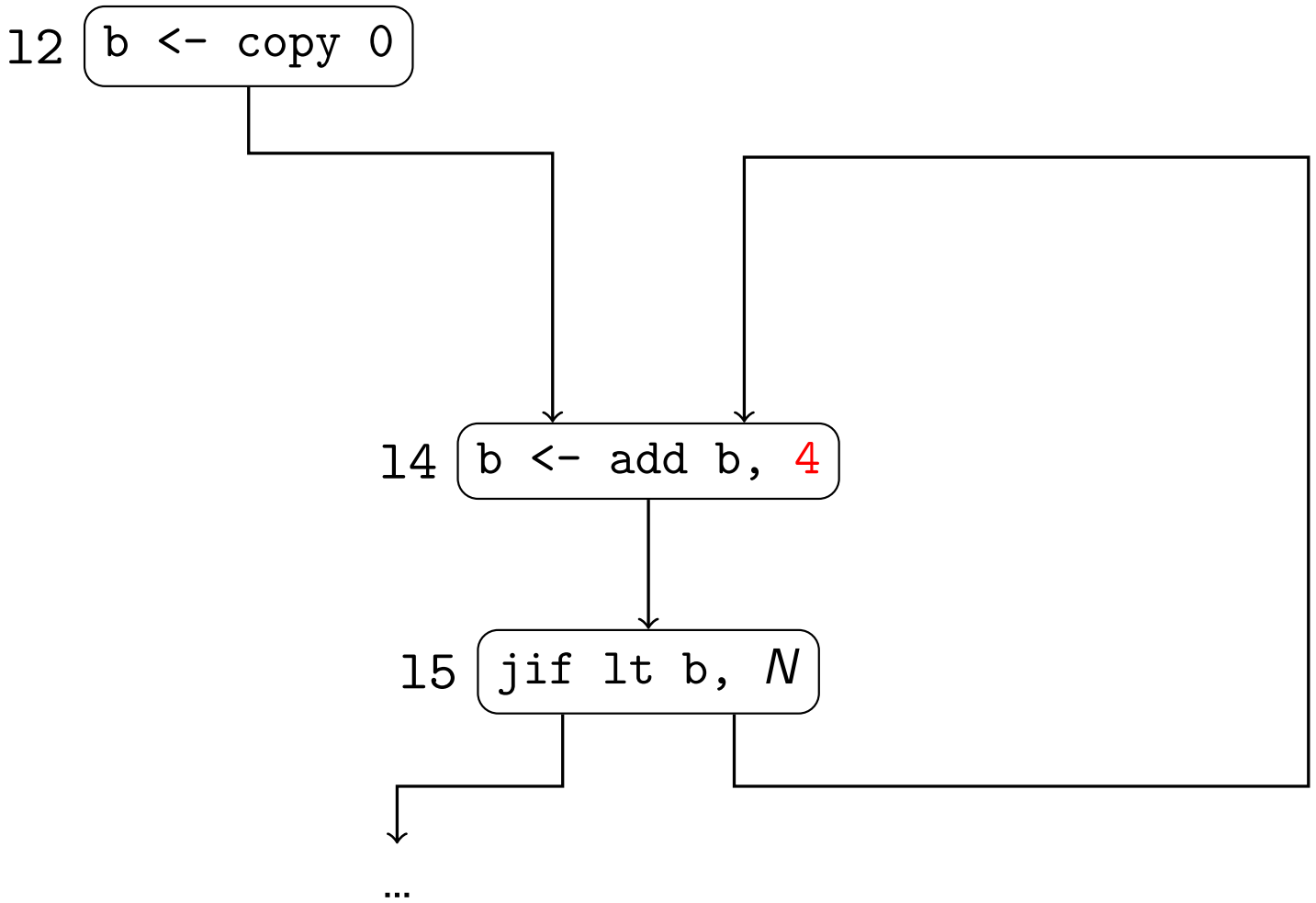
Analyses dataflow au service d'optimisations : CF (2/2)



Analyses dataflow au service d'optimisations : CF (2/2)



Analyses dataflow au service d'optimisations : CF (2/2)



Analyses dataflow au service d'optimisation : CSE (1/2)

L'élimination des sous-expressions communes (*Common-Subexpression Elimination*) factorise des calculs pleinement redondants, i.e., dont le résultat est déjà disponible sur tous les chemins d'exécution.

Analyses dataflow au service d'optimisation : CSE (1/2)

L'élimination des sous-expressions communes (*Common-Subexpression Elimination*) factorise des calculs pleinement redondants, i.e., dont le résultat est déjà disponible sur tous les chemins d'exécution.

Si l est l'instruction disponible à l'adresse l, alors :

- $Gen(l)$ est l'ensemble des expressions calculées par l,
- $Kill(l)$ est l'ensemble des expressions invalidées par l.

Par exemple, si l est `c <- add a, b`, alors :

$$Gen(l) = \{\mathtt{a} + \mathtt{b}\}, \qquad Kill(l) = \{e \in Exp \mid \mathtt{c} \in FV(e)\}.$$

Analyses dataflow au service d'optimisation : CSE (1/2)

L'élimination des sous-expressions communes (*Common-Subexpression Elimination*) factorise des calculs pleinement redondants, i.e., dont le résultat est déjà disponible sur tous les chemins d'exécution.

Si l est l'instruction disponible à l'adresse l, alors :

- $Gen(l)$ est l'ensemble des expressions calculées par l,
- $Kill(l)$ est l'ensemble des expressions invalidées par l.

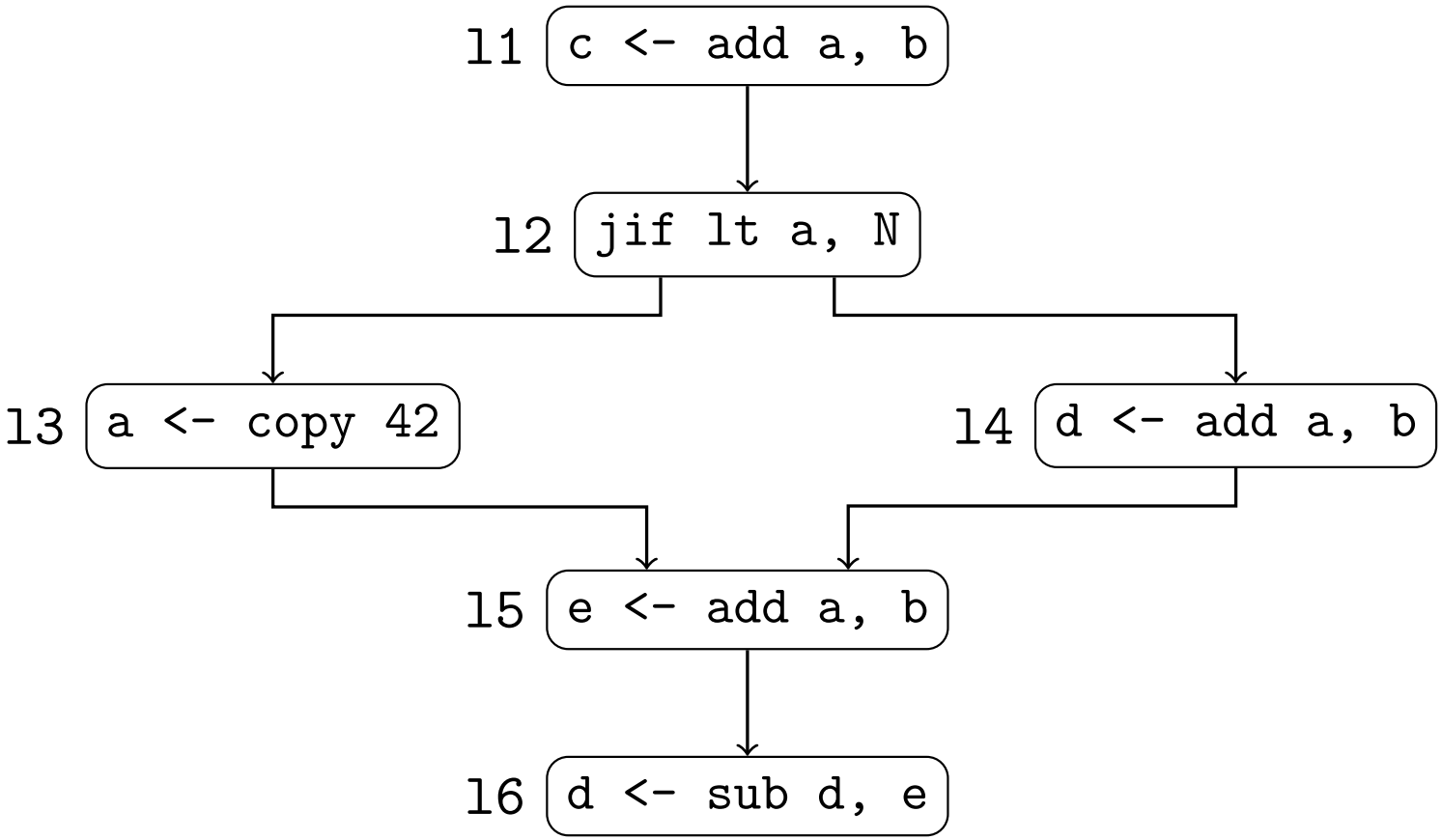
Par exemple, si l est `c <- add a, b`, alors :

$$Gen(l) = \{\mathtt{a} + \mathtt{b}\}, \qquad Kill(l) = \{e \in Exp \mid \mathtt{c} \in FV(e)\}.$$

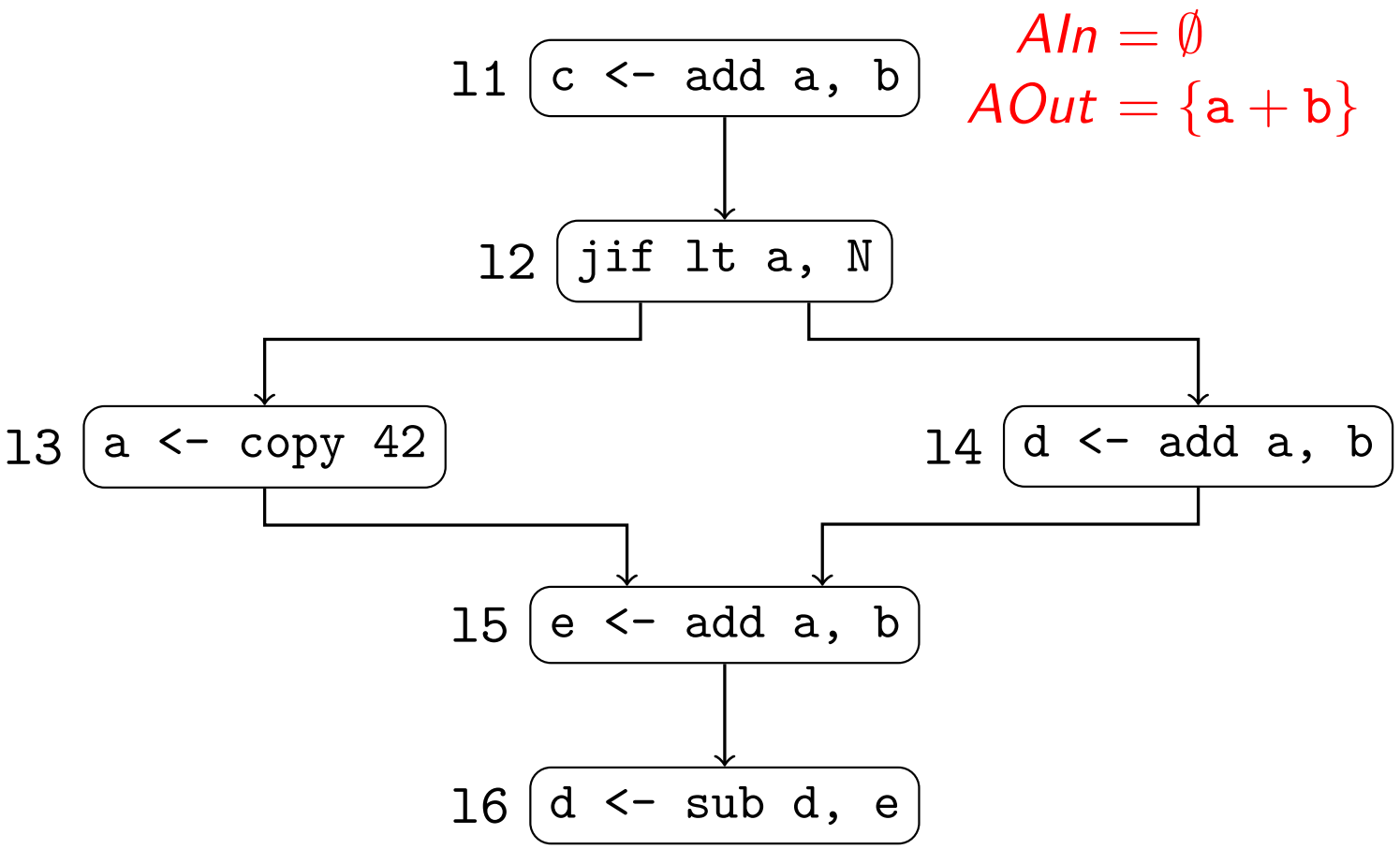
Pour calculer les expressions disponibles, on effectue une analyse dataflow en avant avec $P \triangleq \mathcal{P}(Exp) \times \mathcal{P}(Exp)$.

$$AIn(l) = \bigcap_{l' \in Pred(l)} AOut(l') \qquad AOut(l) = Gen(l) \cup (AIn(l) \setminus Kill(l))$$

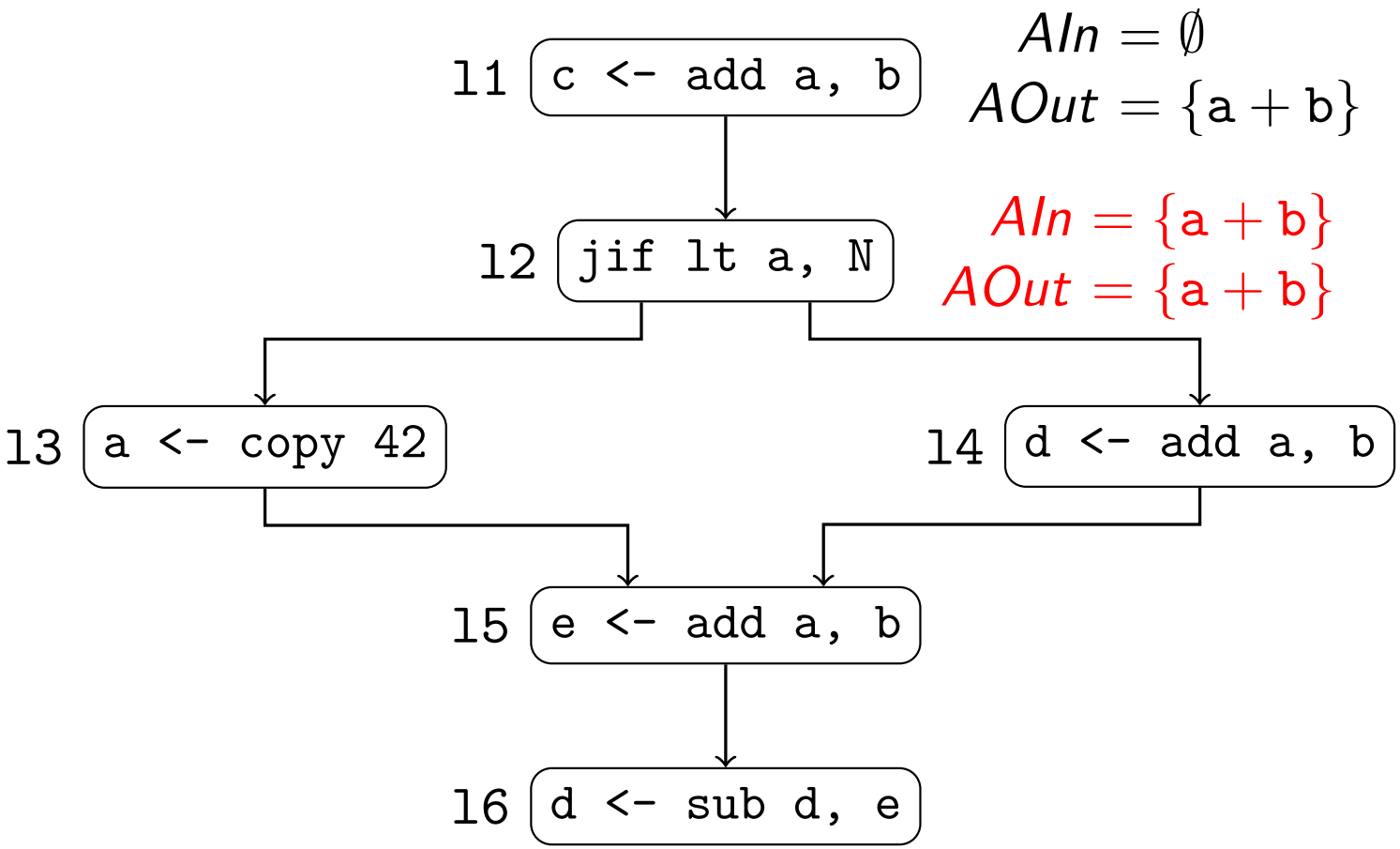
Analyses dataflow au service d'optimisation : CSE (2/2)



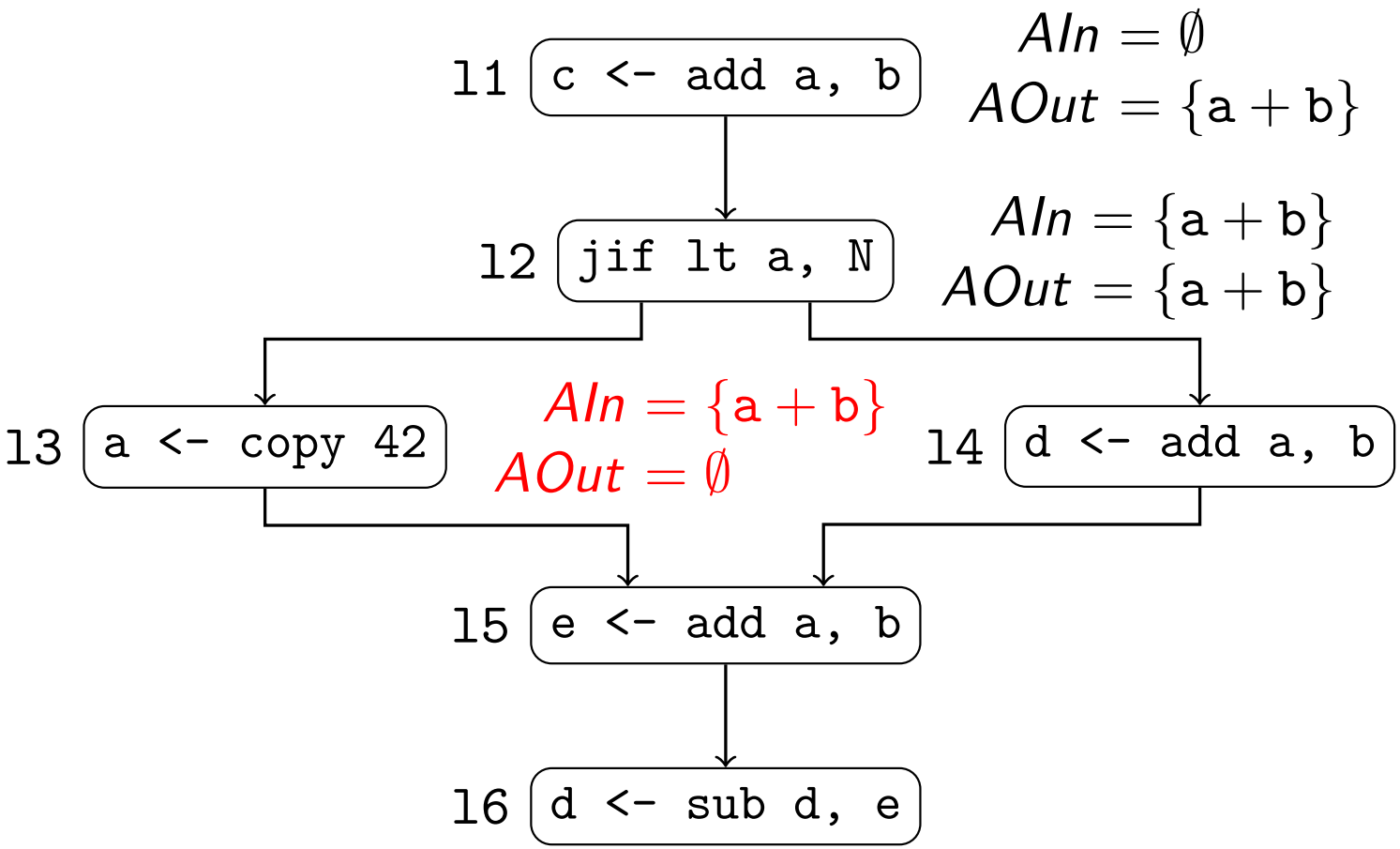
Analyses dataflow au service d'optimisation : CSE (2/2)



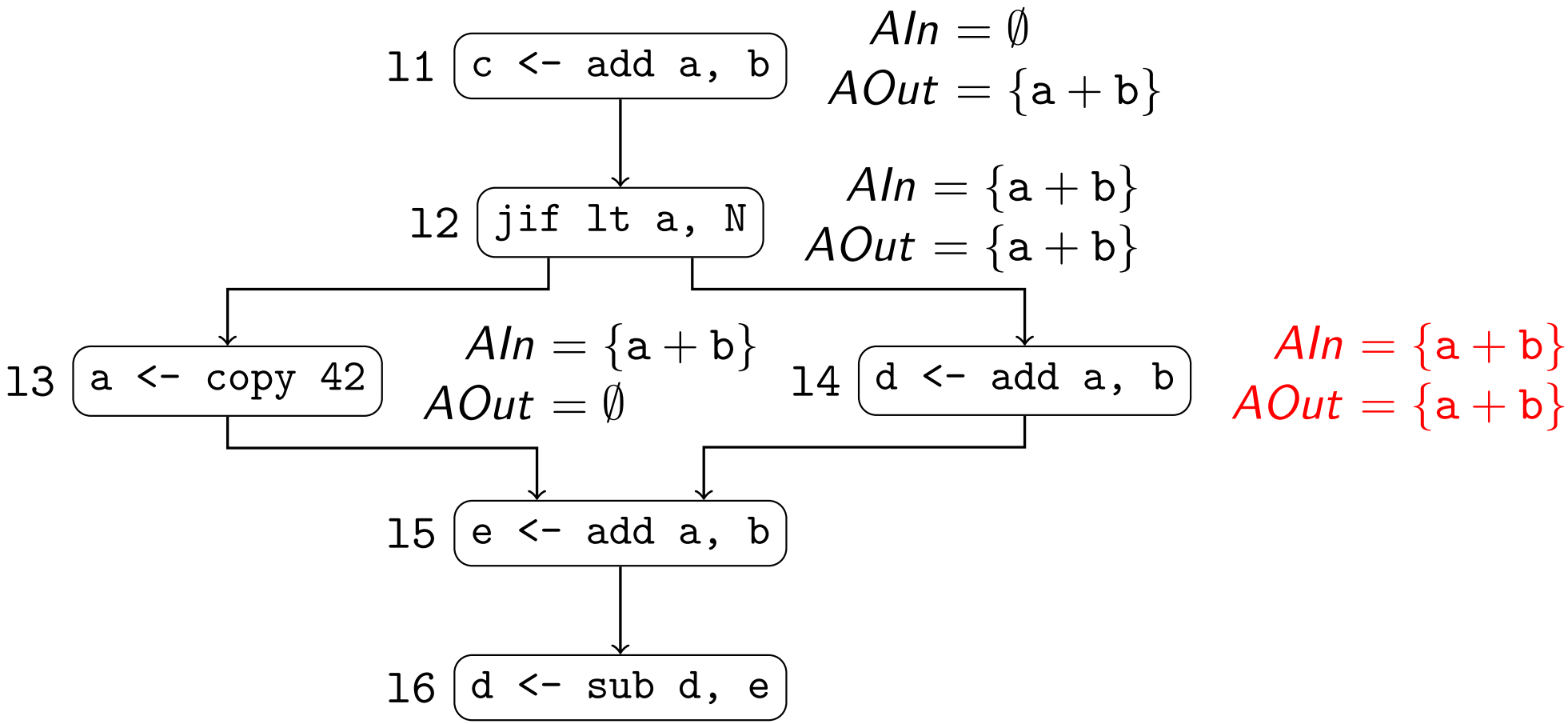
Analyses dataflow au service d'optimisation : CSE (2/2)



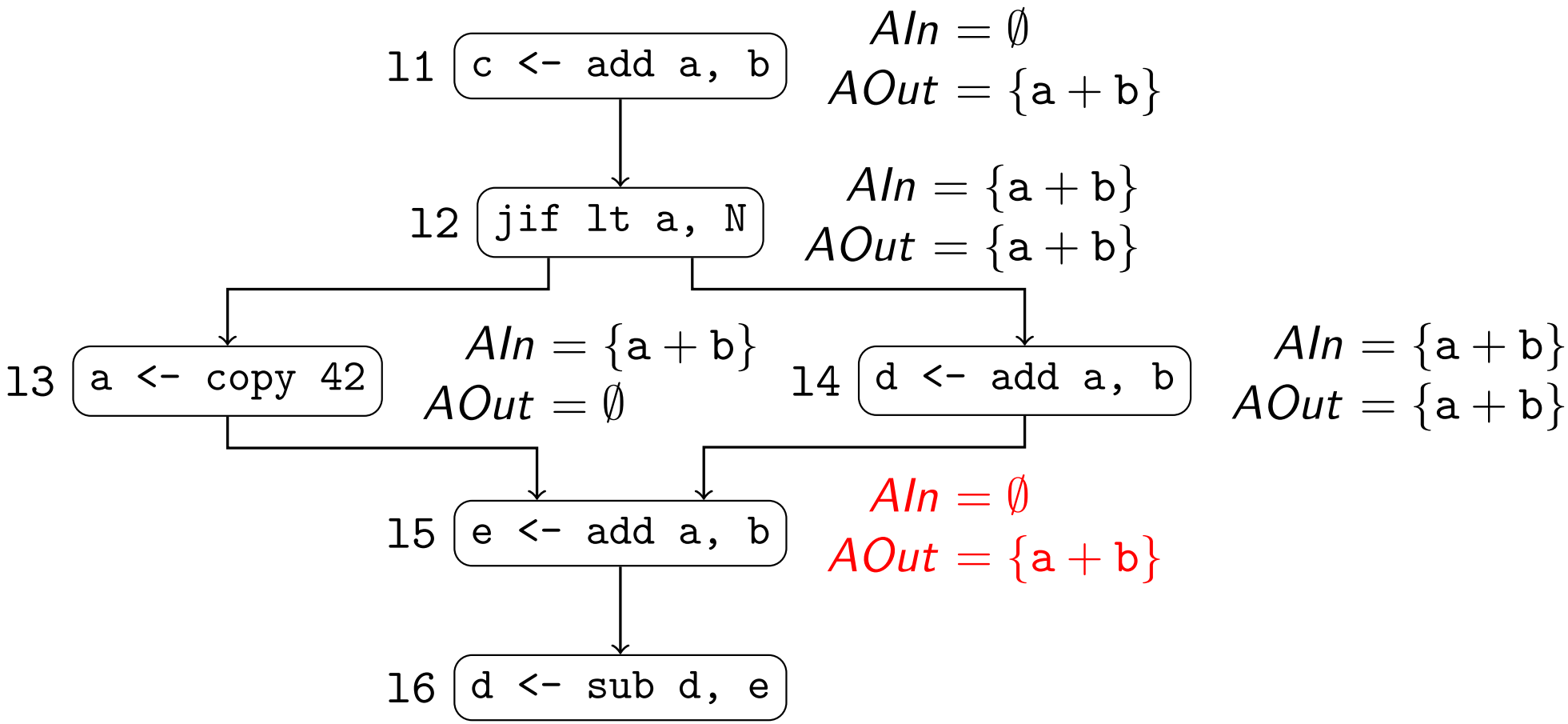
Analyses dataflow au service d'optimisation : CSE (2/2)



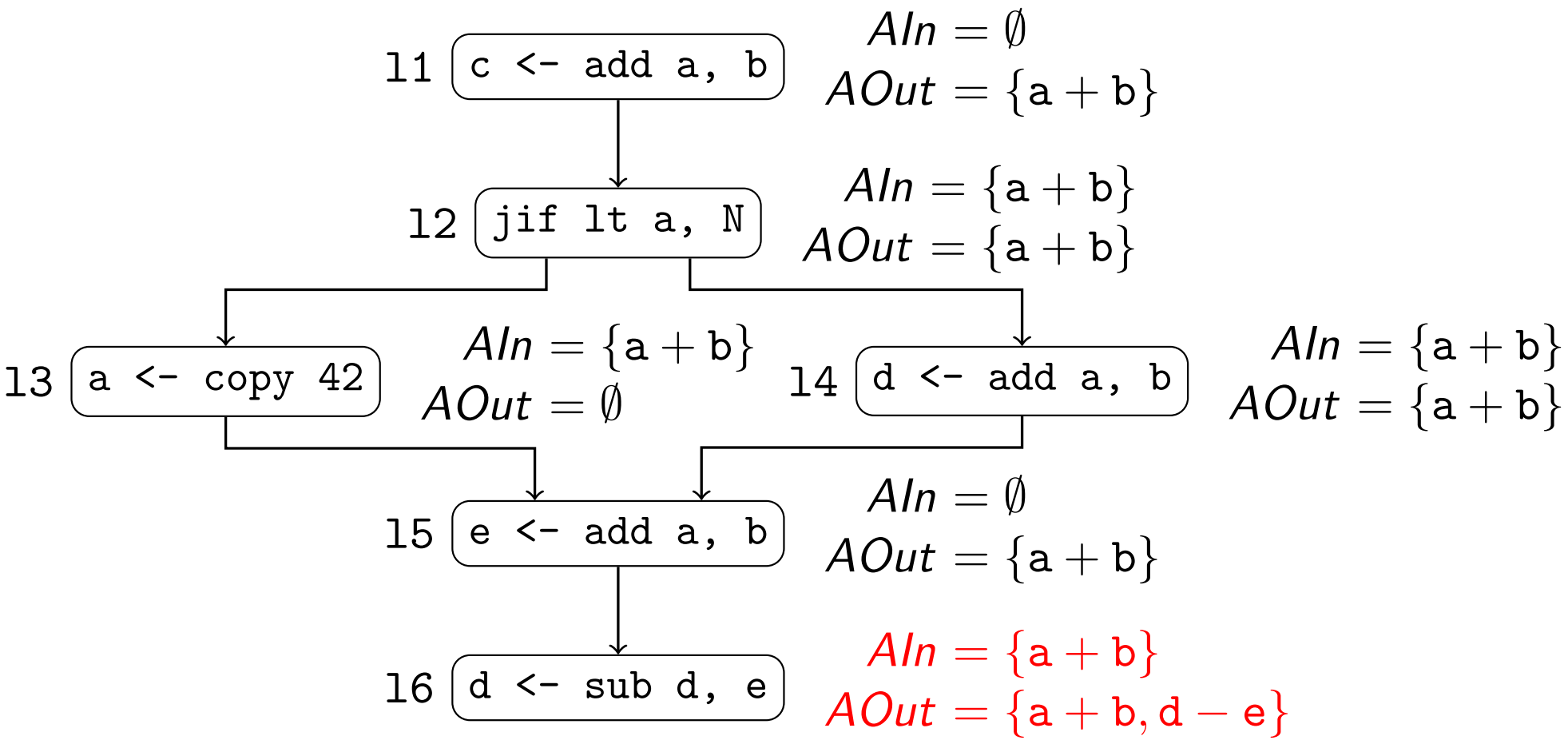
Analyses dataflow au service d'optimisation : CSE (2/2)



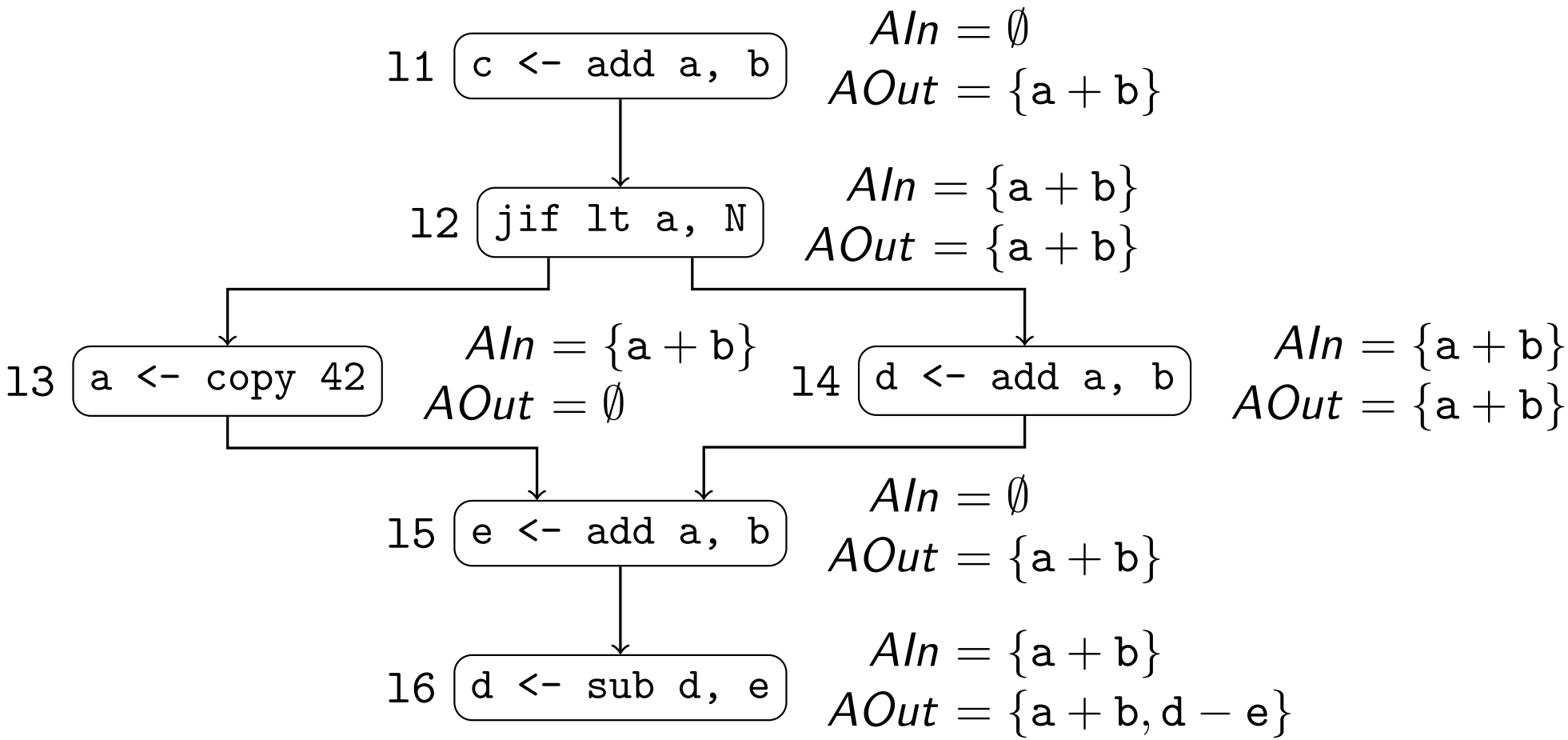
Analyses dataflow au service d'optimisation : CSE (2/2)



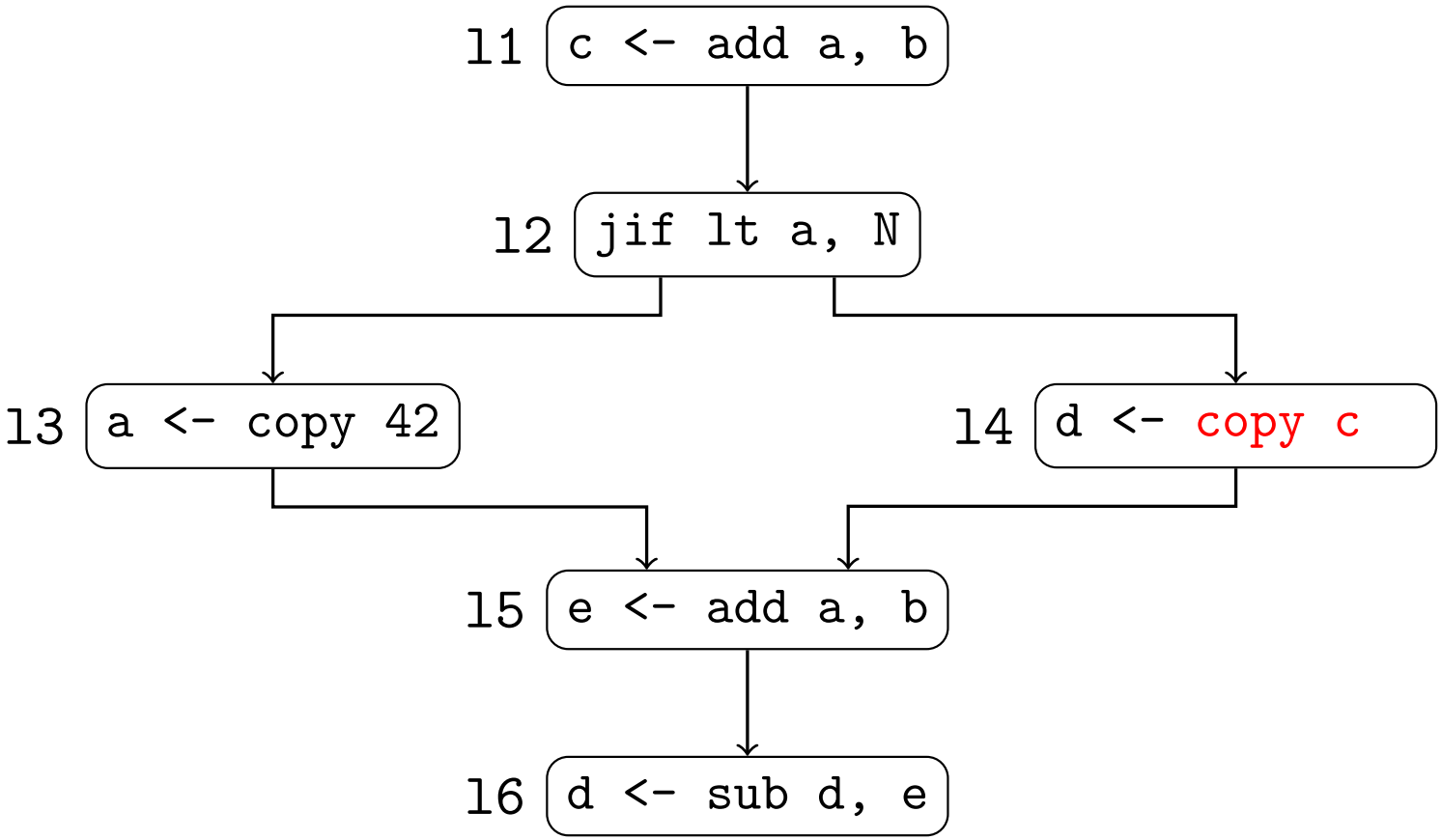
Analyses dataflow au service d'optimisation : CSE (2/2)



Analyses dataflow au service d'optimisation : CSE (2/2)



Analyses dataflow au service d'optimisation : CSE (2/2)



Outline

Au delà du cadre classique

La litérature regorge d'autres analyses et optimisations, dont beaucoup sont implémentées dans les compilateurs industriels.

Dans quelles grandes directions développer le cadre présenté jusqu'ici ?

Au delà du cadre classique

La litérature regorge d'autres analyses et optimisations, dont beaucoup sont implémentées dans les compilateurs industriels.

Dans quelles grandes directions développer le cadre présenté jusqu'ici ?

- Améliorer la **précision** des analyses.
  - Des espaces de propriétés beaucoup plus riches.
    - e.g., domaines abstraits numériques, des intervalles aux polyèdres.
  - Des méthodes de calcul de point fixe plus sophistiquées.
    - e.g., partitionnement de trace ou itération de politique.

Au delà du cadre classique

La litérature regorge d'autres analyses et optimisations, dont beaucoup sont implémentées dans les compilateurs industriels.

Dans quelles grandes directions développer le cadre présenté jusqu'ici ?

- Améliorer la **précision** des analyses.
  - Des espaces de propriétés beaucoup plus riches.
    - e.g., domaines abstraits numériques, des intervalles aux polyèdres.
  - Des méthodes de calcul de point fixe plus sophistiquées.
    - e.g., partitionnement de trace ou itération de politique.

$\Rightarrow$ analyse statique par interprétation abstraite, cf. cours de master 2.

Au delà du cadre classique

La litérature regorge d'autres analyses et optimisations, dont beaucoup sont implémentées dans les compilateurs industriels.

Dans quelles grandes directions développer le cadre présenté jusqu'ici ?

- Améliorer la **précision** des analyses.
  - Des espaces de propriétés beaucoup plus riches.
    - e.g., domaines abstraits numériques, des intervalles aux polyèdres.
  - Des méthodes de calcul de point fixe plus sophistiquées.
    - e.g., partitionnement de trace ou itération de politique.

  $\Rightarrow$ analyse statique par interprétation abstraite, cf. cours de master 2.
- Améliorer la **performance** des analyses.
  - Astuces d'implémentation, e.g., itération chaotique, vecteurs de bits...
  - Meilleures représentations intermédiaires.

Au delà du cadre classique

La litérature regorge d'autres analyses et optimisations, dont beaucoup sont implémentées dans les compilateurs industriels.

Dans quelles grandes directions développer le cadre présenté jusqu'ici ?

- Améliorer la **précision** des analyses.
  - Des espaces de propriétés beaucoup plus riches.
    - e.g., domaines abstraits numériques, des intervalles aux polyèdres.
  - Des méthodes de calcul de point fixe plus sophistiquées.
    - e.g., partitionnement de trace ou itération de politique.

  $\Rightarrow$ analyse statique par interprétation abstraite, cf. cours de master 2.
- Améliorer la **performance** des analyses.
  - Astuces d'implémentation, e.g., itération chaotique, vecteurs de bits...
  - Meilleures représentations intermédiaires.

  $\Rightarrow$ compilation des programmes en forme à assignation statique unique (*static single assignment form*), cf. le reste de la séance.

Vers la forme SSA

Les analyses dataflow vues jusqu'ici associent des propriétés aux points du programme. Pourquoi ?

Vers la forme SSA

Les analyses dataflow vues jusqu'ici associent des propriétés aux points du programme. Pourquoi ? Parce que nos variables sont trop impératives !

Vers la forme SSA

Les analyses dataflow vues jusqu'ici associent des propriétés aux points du programme. Pourquoi ? Parce que nos variables sont trop impératives !

Un programme en forme SSA n'a qu'**une seule définition par variable**.

Vers la forme SSA

Les analyses dataflow vues jusqu'ici associent des propriétés aux points du programme. Pourquoi ? Parce que nos variables sont trop impératives !

Un programme en forme SSA n'a qu'**une seule définition par variable**.

```
l0: x <- add a, b
l1: a <- copy 12
l2: x <- mul a, 2
l3: b <- copy x
```

SSA ×

Vers la forme SSA

Les analyses dataflow vues jusqu'ici associent des propriétés aux points du programme. Pourquoi ? Parce que nos variables sont trop impératives !

Un programme en forme SSA n'a qu'**une seule définition par variable**.

```
l0: x <- add a, b
l1: a <- copy 12
l2: x <- mul a, 2
l3: b <- copy x
```

SSA ×

```
l0: x0 <- add a0, b0
l1: a1 <- copy 12
l2: x1 <- mul a1, 2
l3: b0 <- copy x1
```

SSA ✓

Vers la forme SSA

Les analyses dataflow vues jusqu'ici associent des propriétés aux points du programme. Pourquoi ? Parce que nos variables sont trop impératives !

Un programme en forme SSA n'a qu'**une seule définition par variable**.

```
l0: x <- add a, b
l1: a <- copy 12
l2: x <- mul a, 2
l3: b <- copy x
```

SSA ×

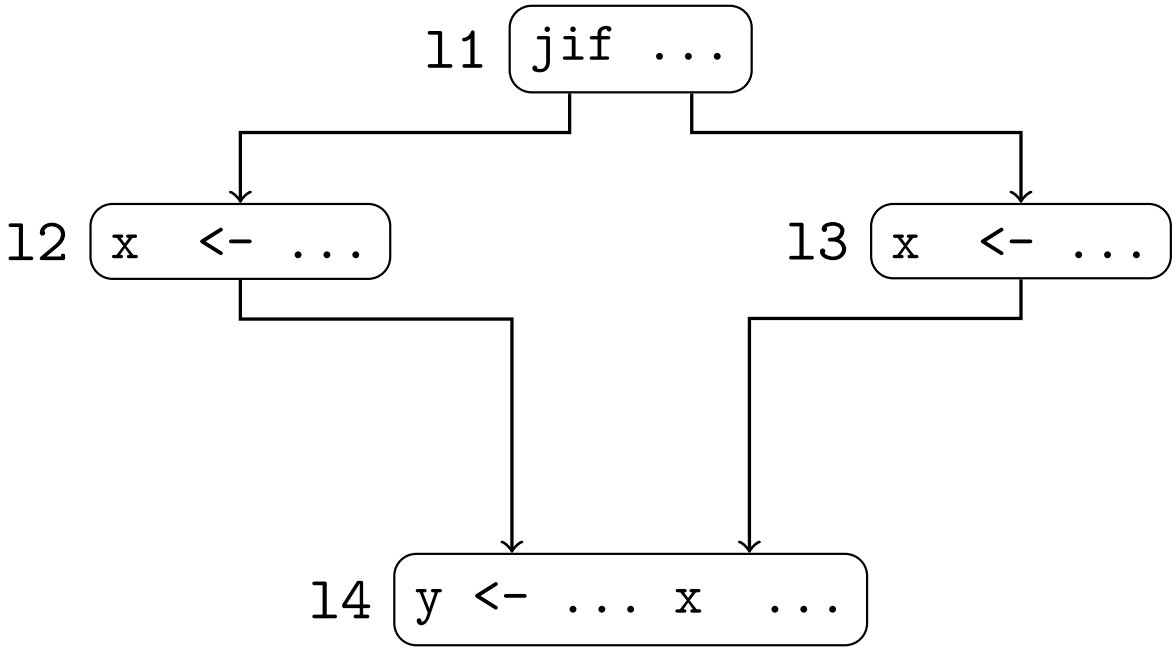
```
l0: x0 <- add a0, b0
l1: a1 <- copy 12
l2: x1 <- mul a1, 2
l3: b0 <- copy x1
```

SSA ✓

Mettre du code en forme SSA exige donc de renommer des variables.
Mais cela ne suffit pas en présence de flot de contrôle...

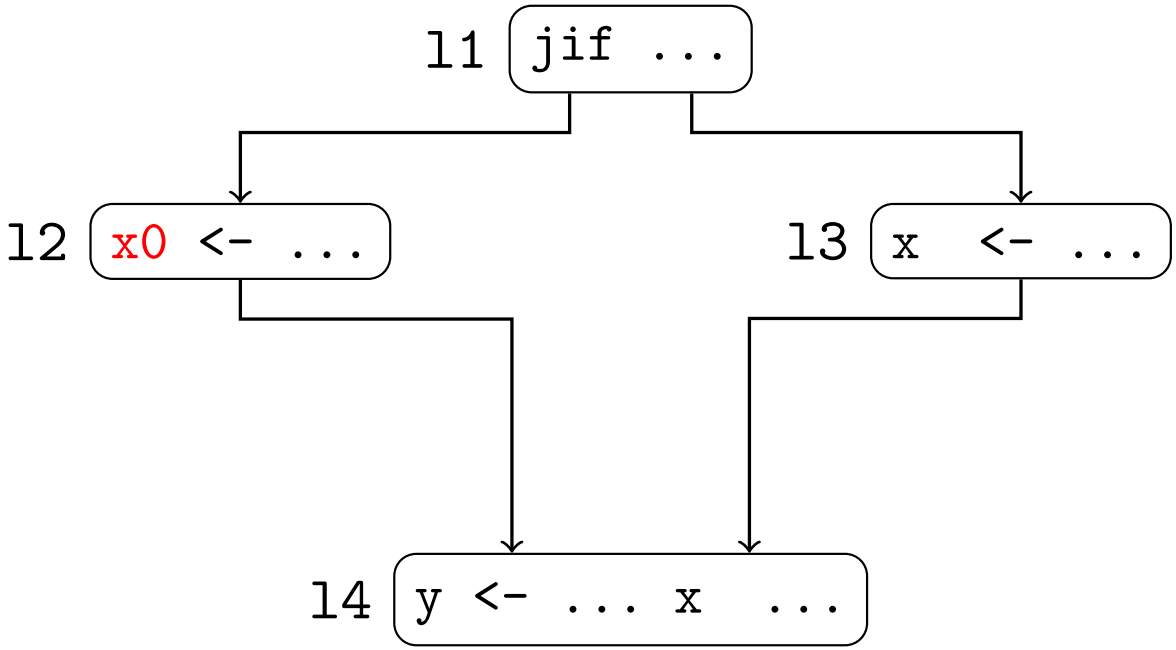
Les fonctions ϕ

Comment mettre en forme SSA le fragment de programme suivant ?



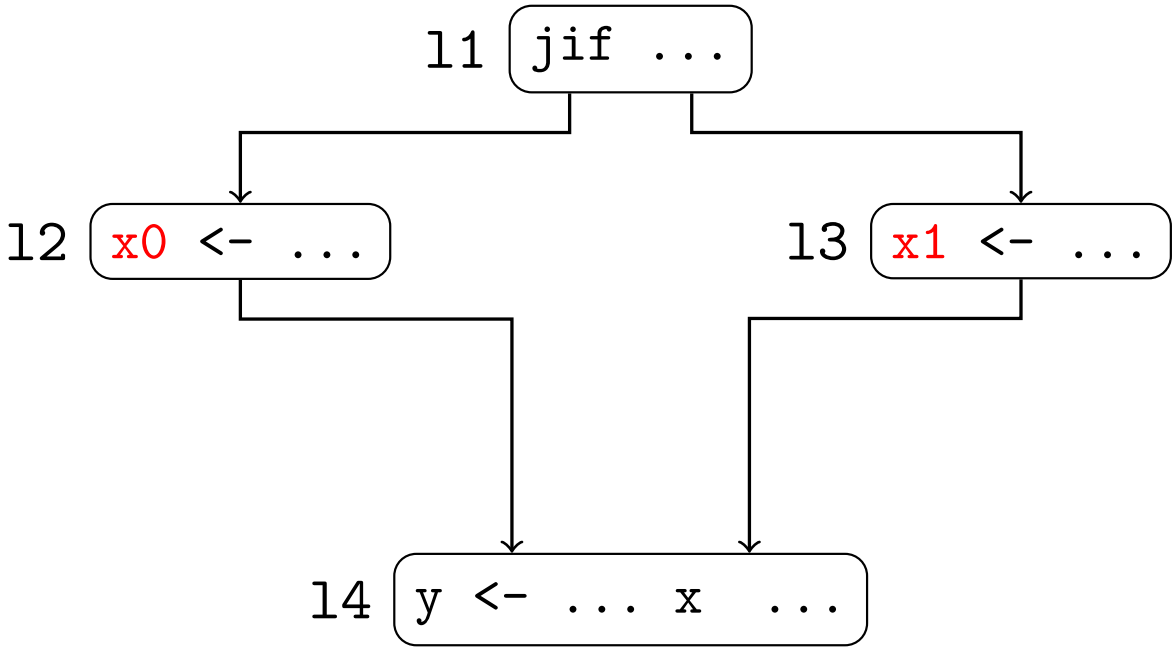
Les fonctions ϕ

Comment mettre en forme SSA le fragment de programme suivant ?



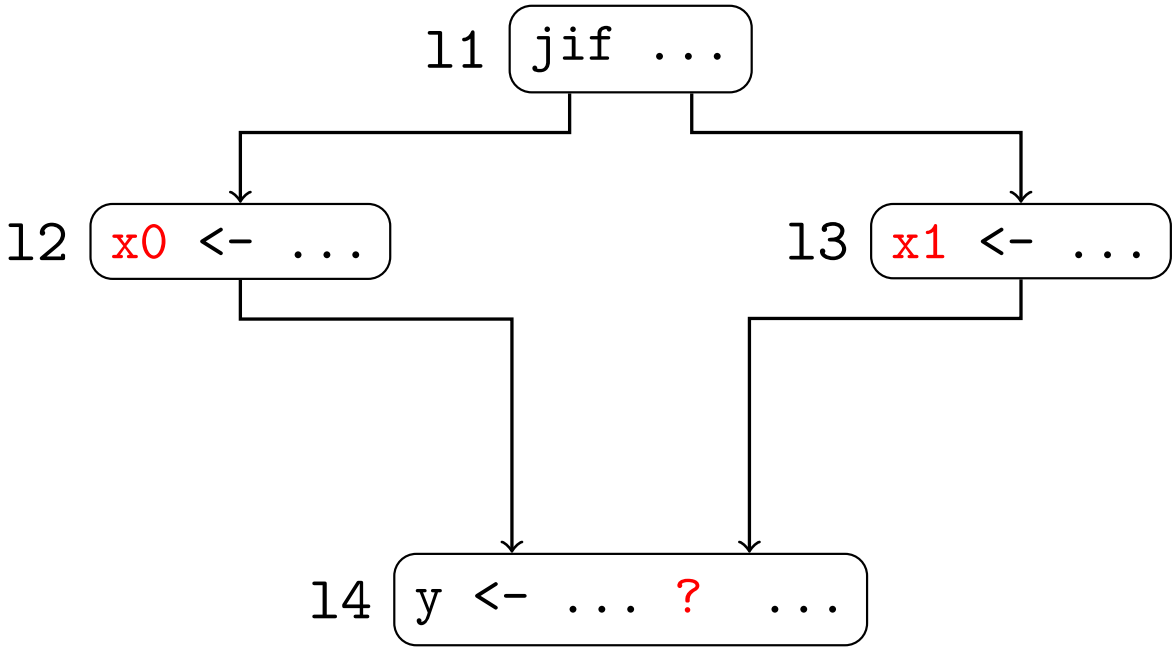
Les fonctions ϕ

Comment mettre en forme SSA le fragment de programme suivant ?



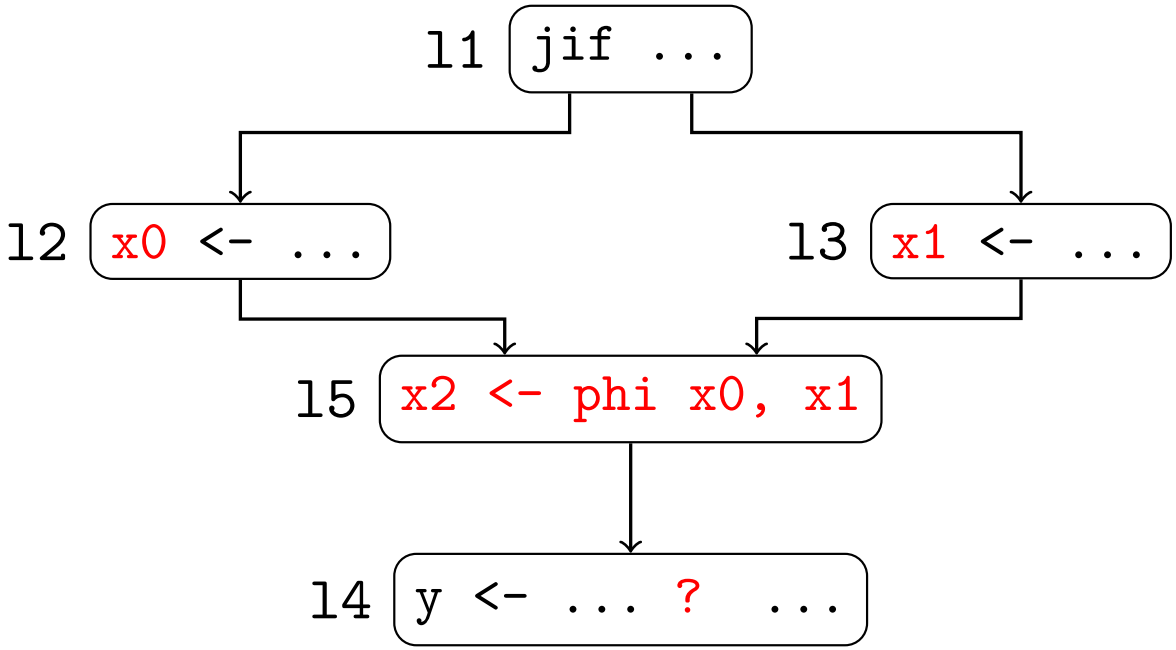
Les fonctions ϕ

Comment mettre en forme SSA le fragment de programme suivant ?



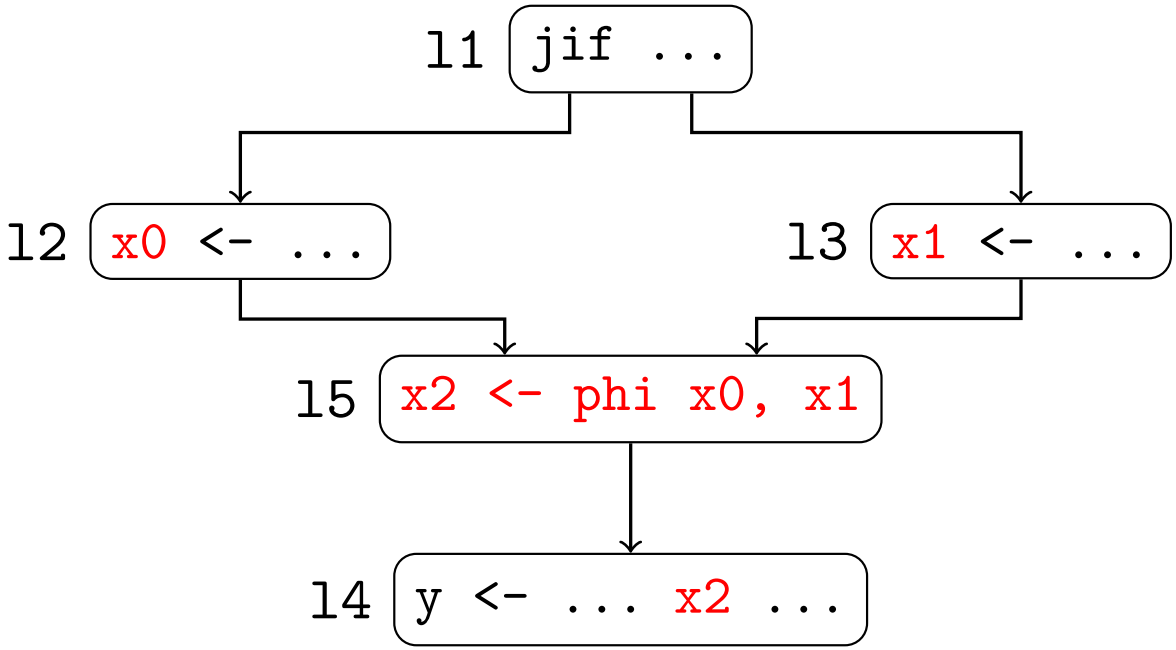
Les fonctions ϕ

Comment mettre en forme SSA le fragment de programme suivant ?



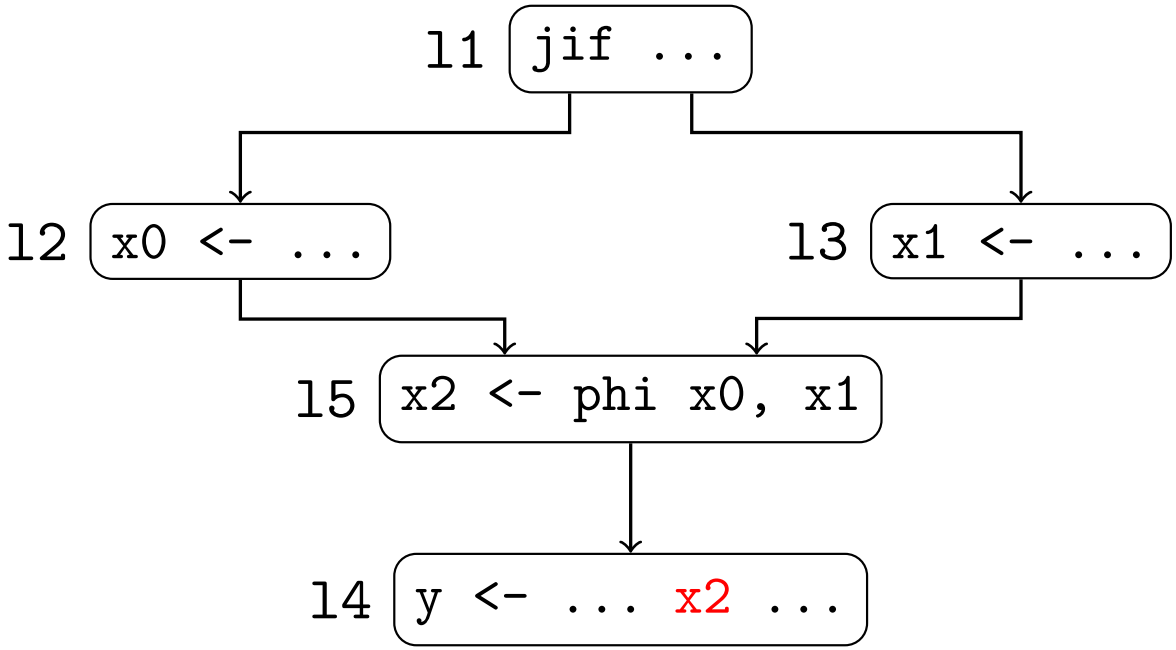
Les fonctions ϕ

Comment mettre en forme SSA le fragment de programme suivant ?



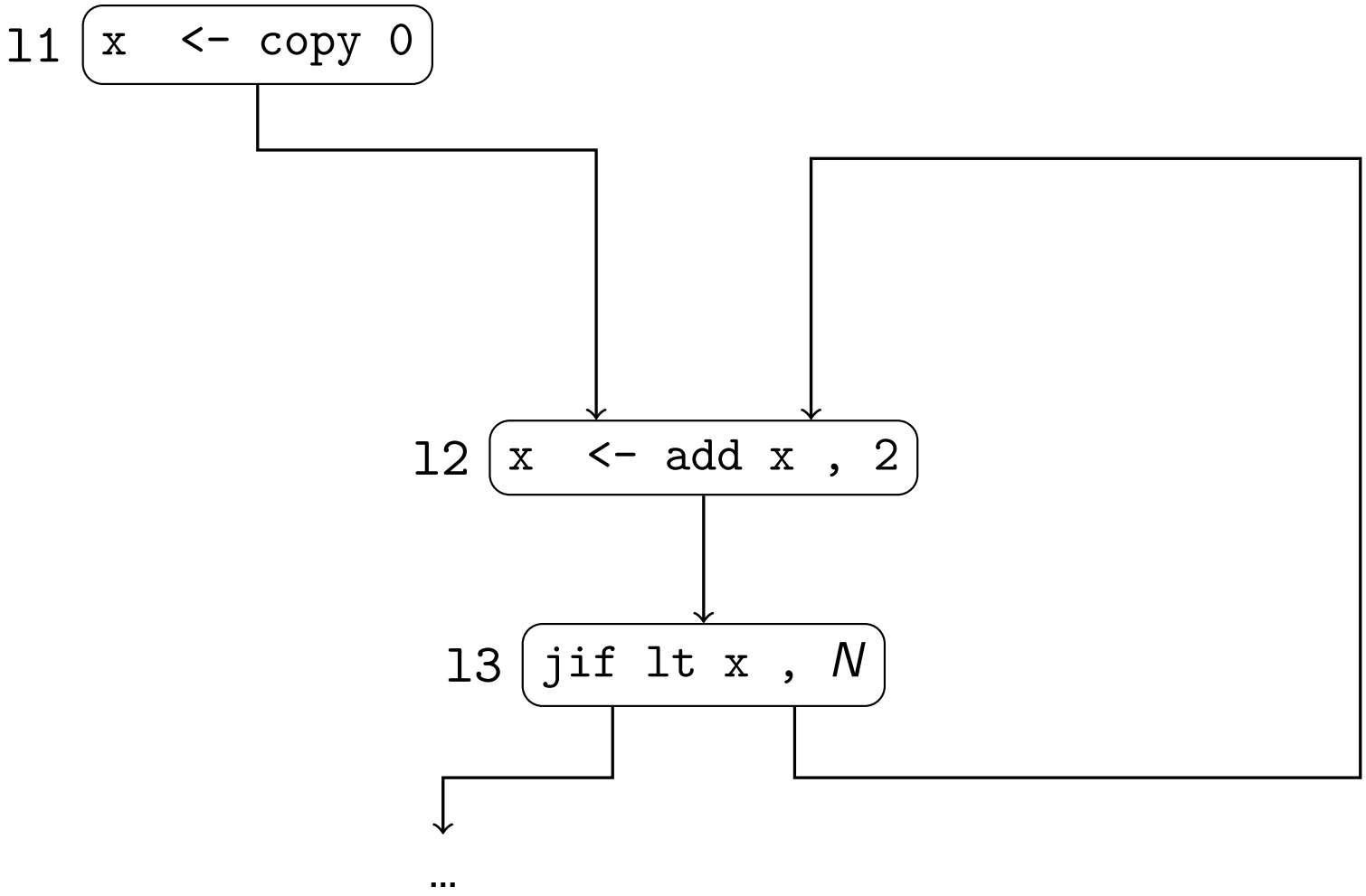
Les fonctions ϕ

Comment mettre en forme SSA le fragment de programme suivant ?

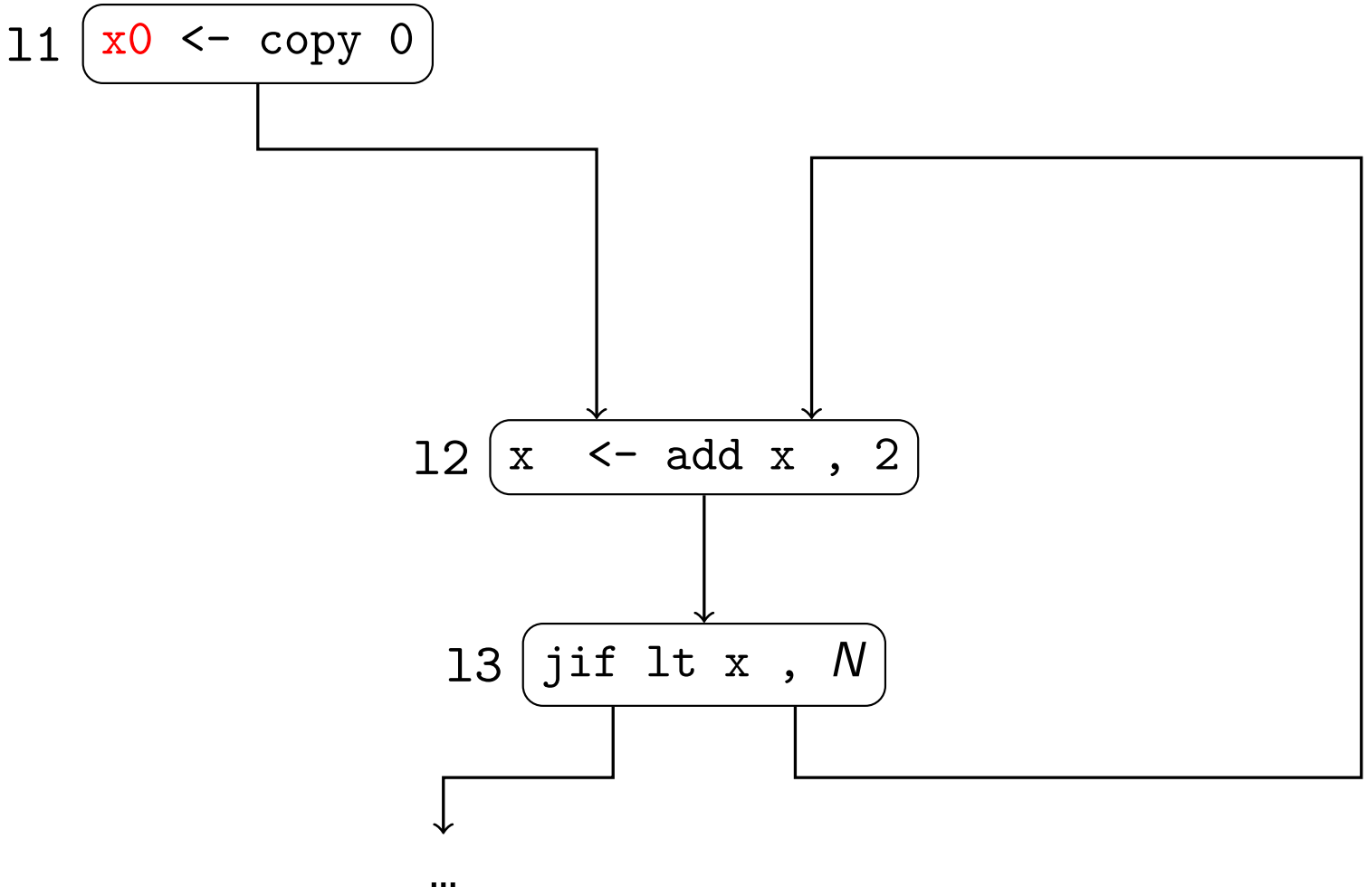


- Les fonctions ϕ (pour “fausses”, ou *phoney*) choisissent quel argument renvoyer en fonction de la provenance du contrôle.
- Elles sont insérées lors de la mise en forme SSA.

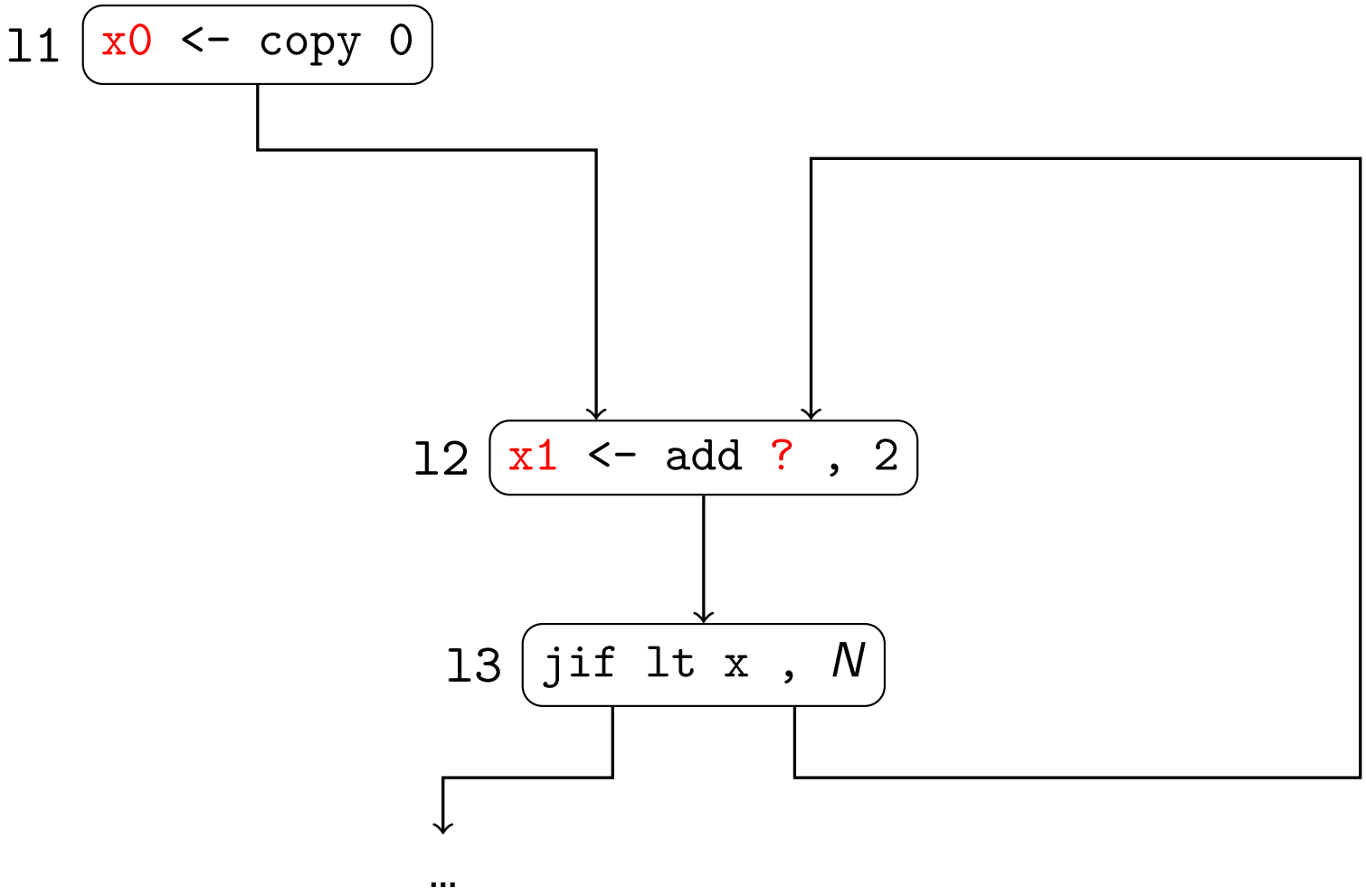
Forme SSA : staticité et dominance



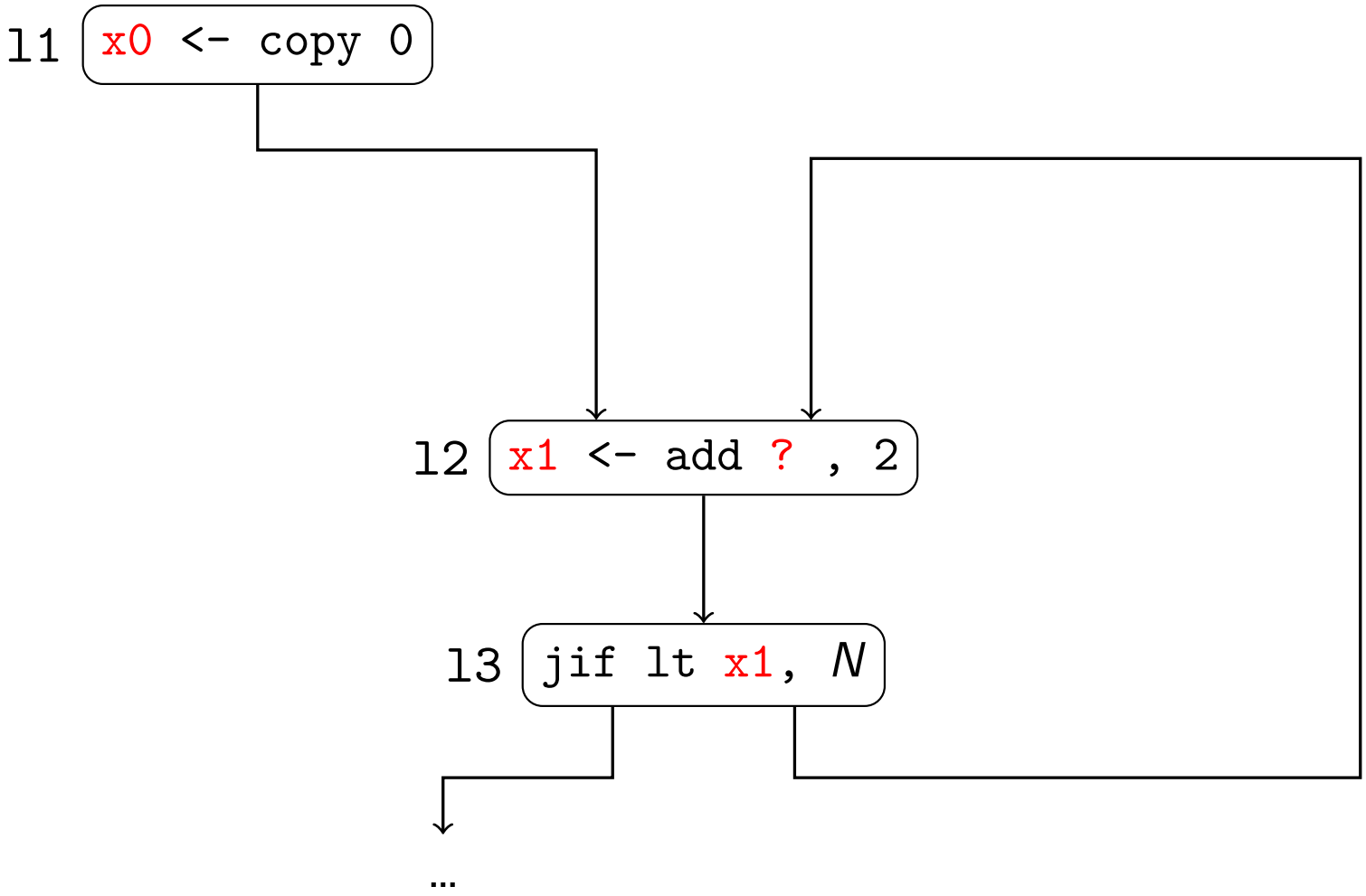
Forme SSA : staticité et dominance



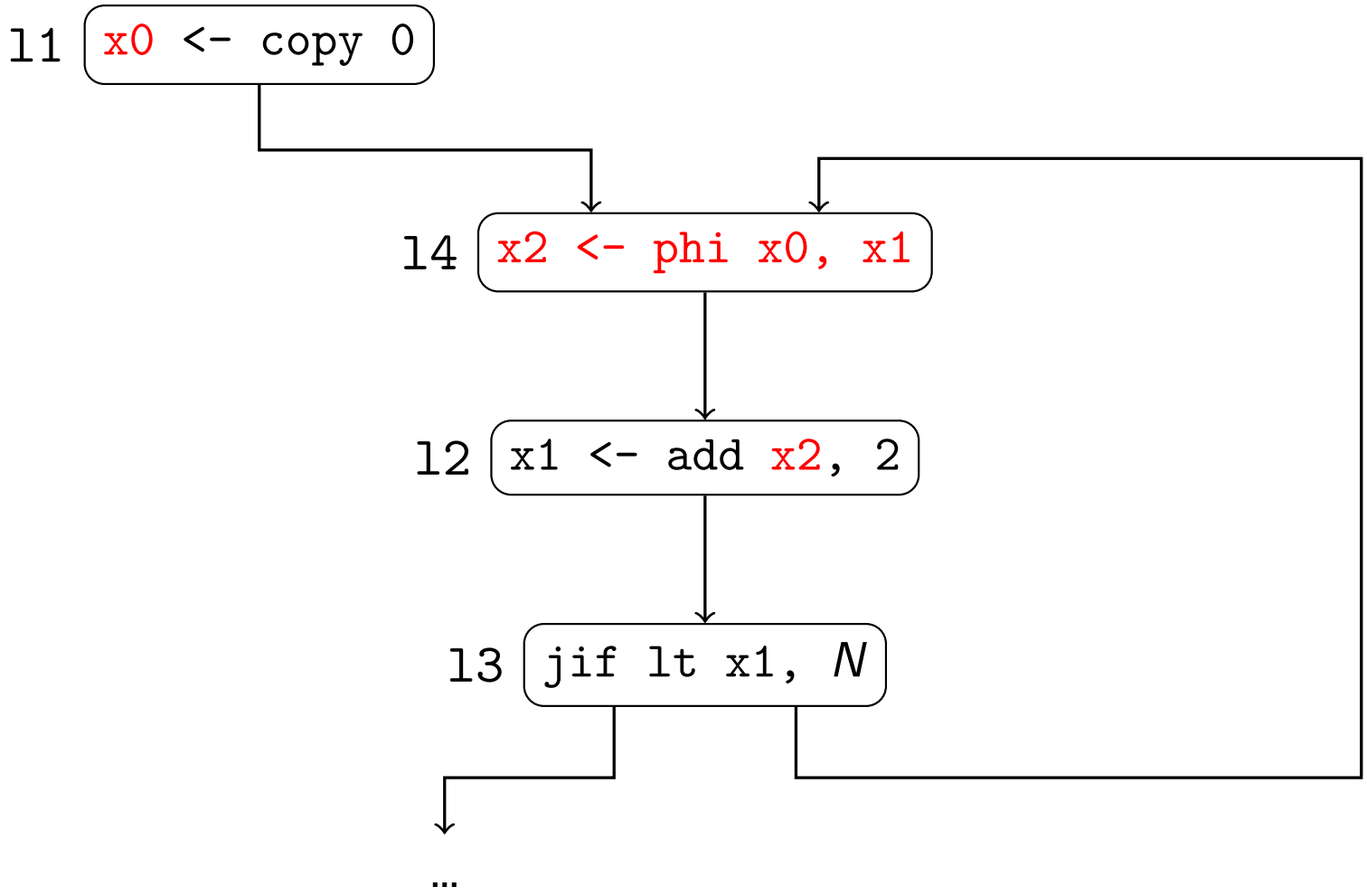
Forme SSA : staticité et dominance



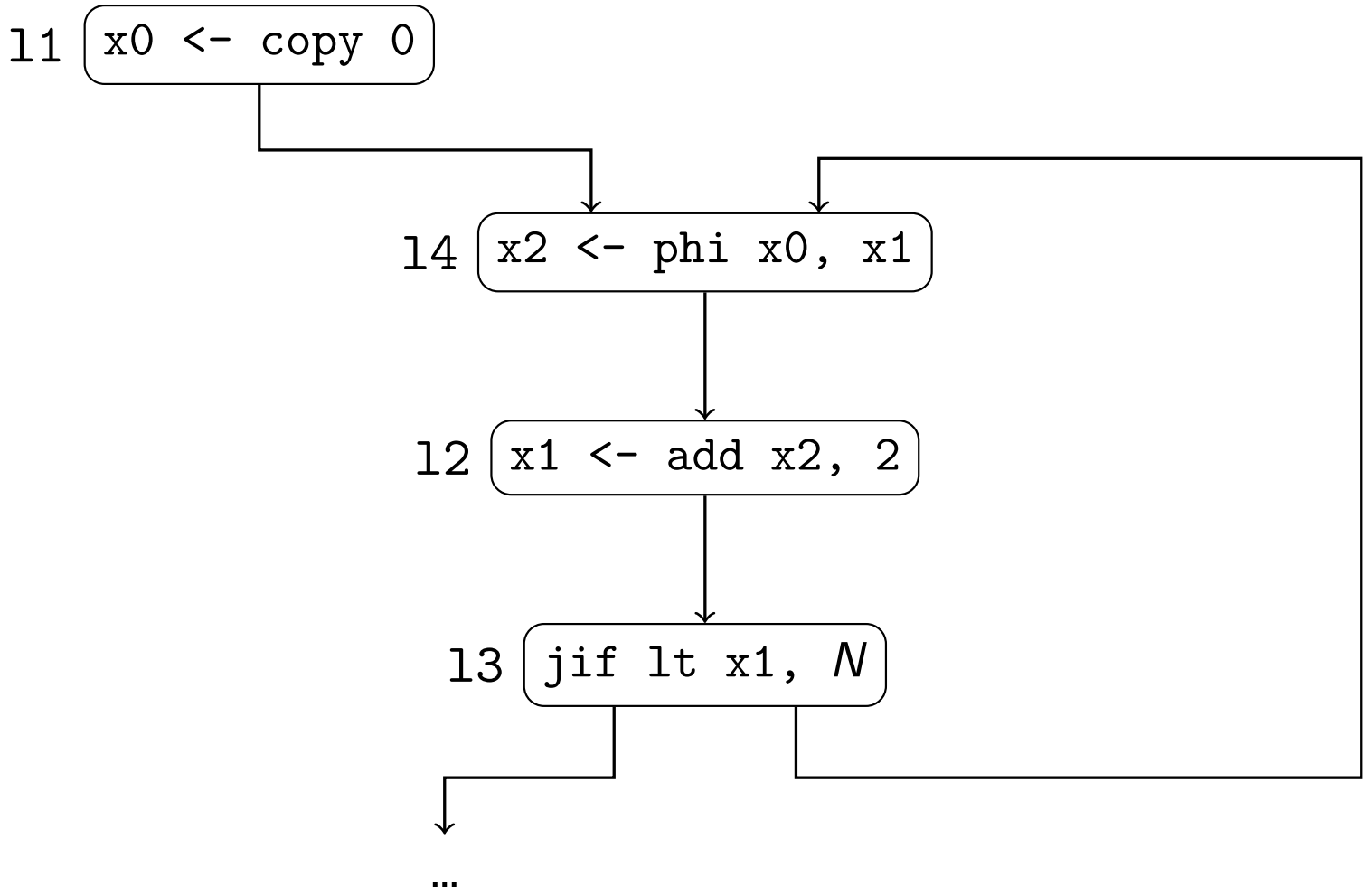
Forme SSA : staticité et dominance



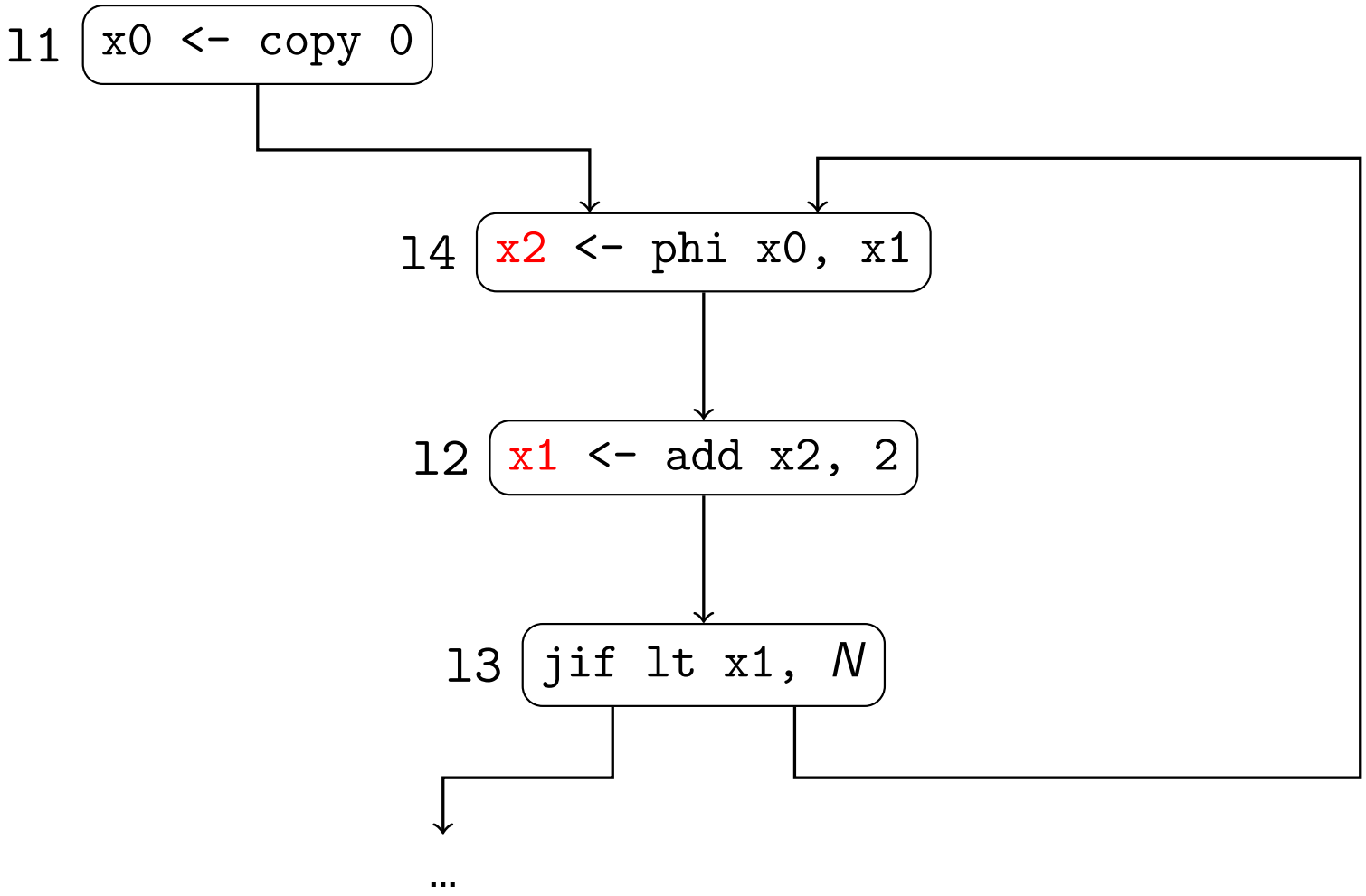
Forme SSA : staticité et dominance



Forme SSA : staticité et dominance

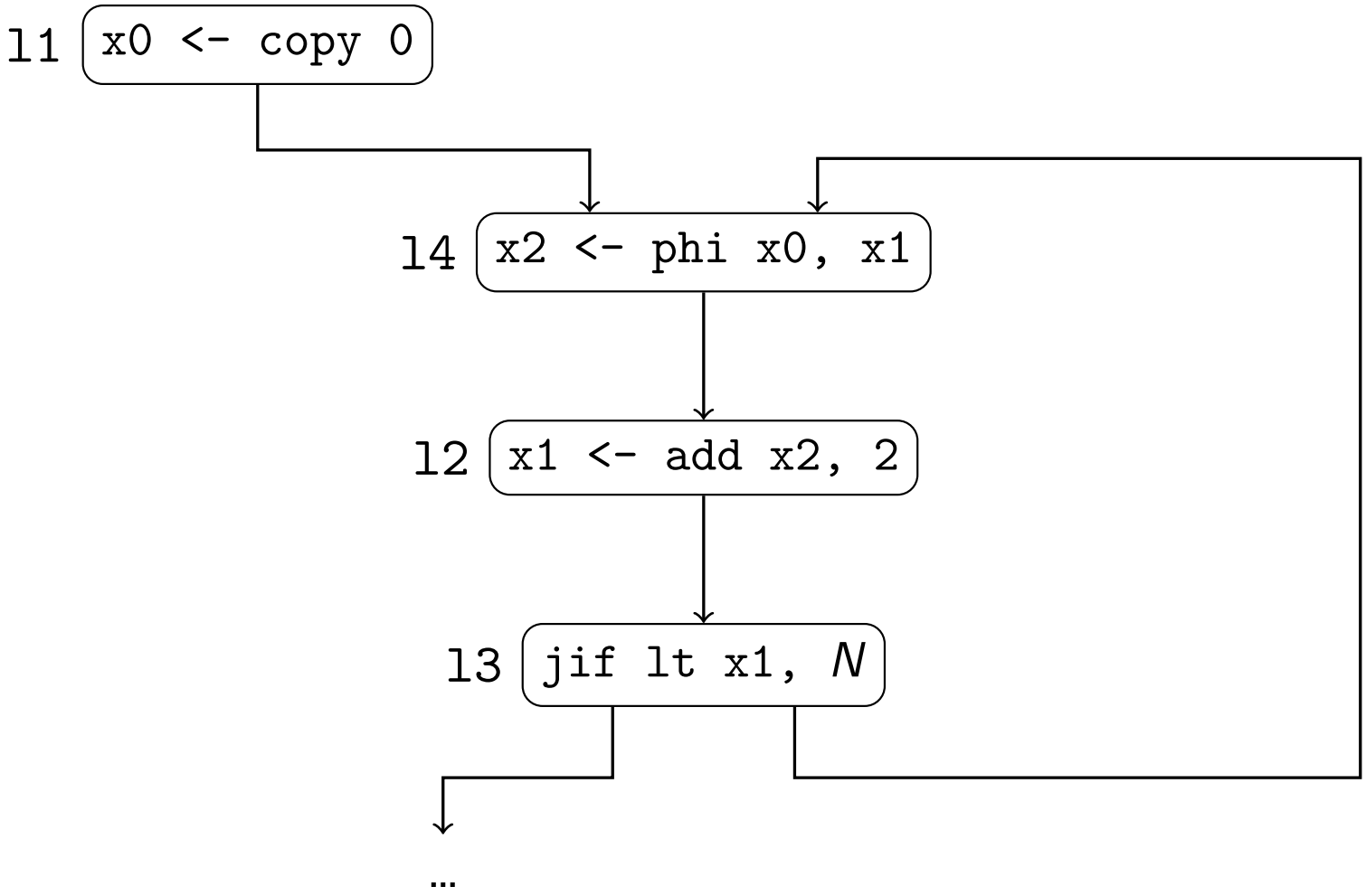


Forme SSA : staticité et dominance



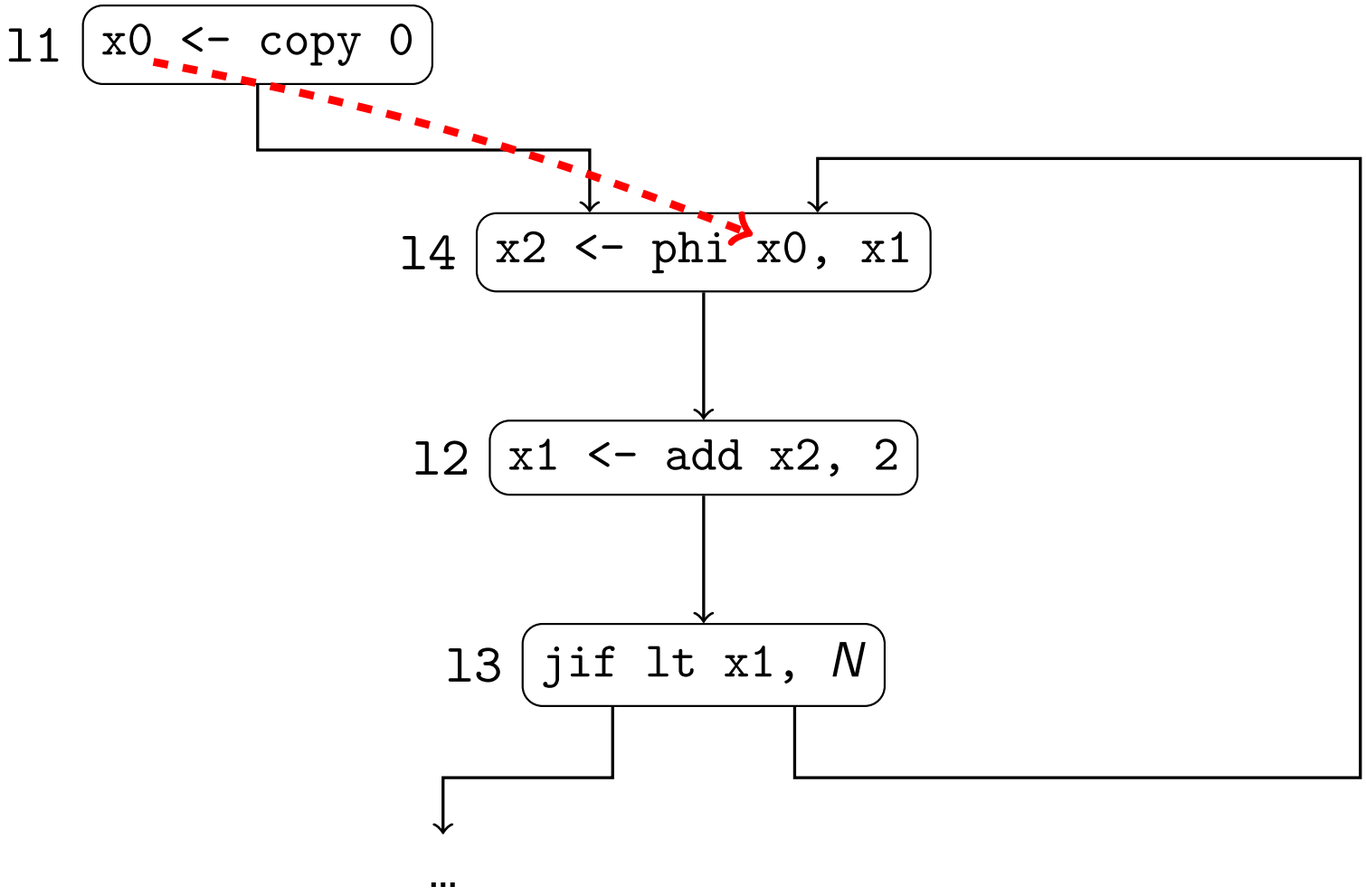
- L'unicité des définitions est *statique*, pas *dynamique* (cf. `x2` et `x1`).

Forme SSA : staticité et dominance



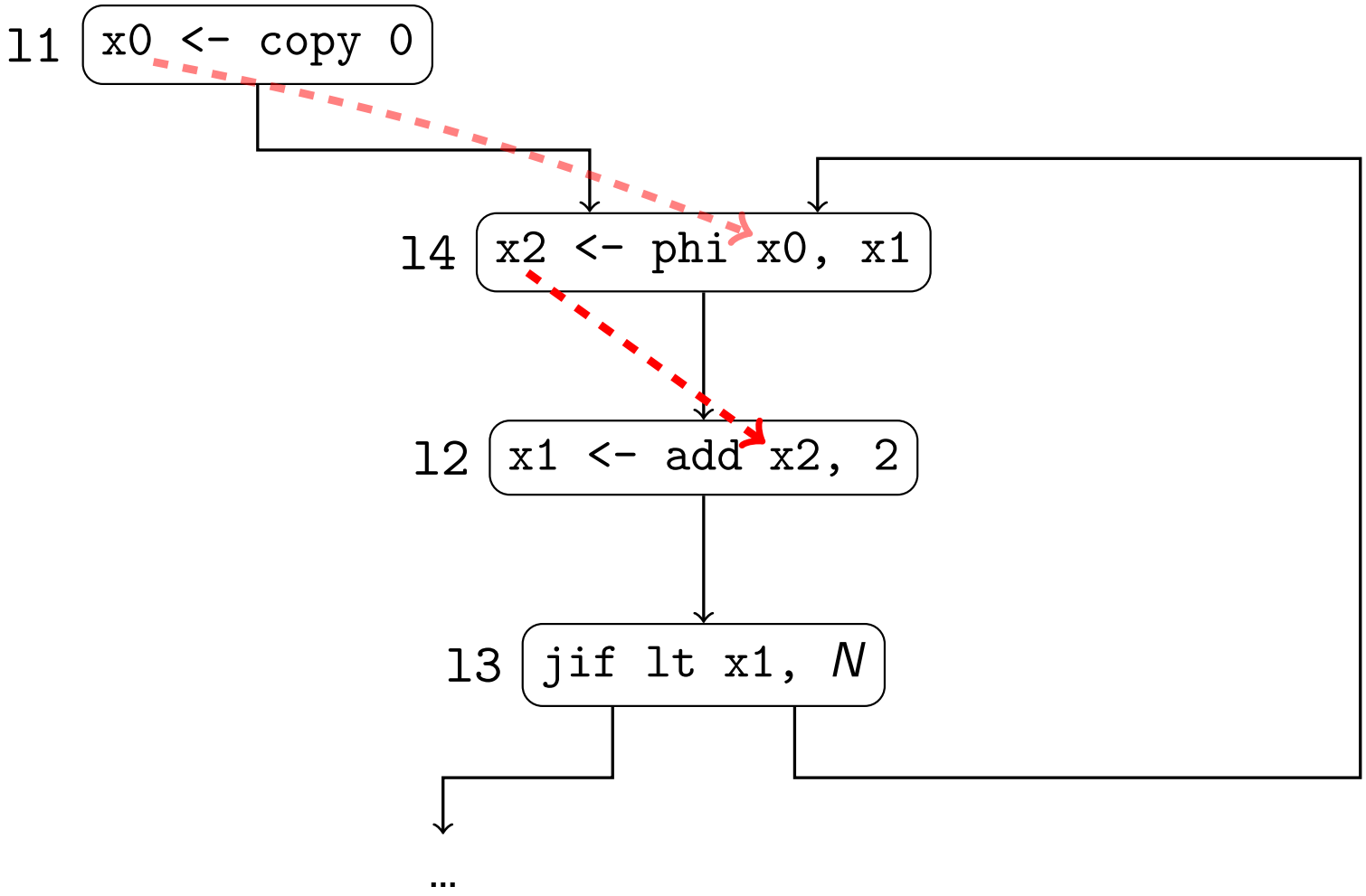
- L'unicité des définitions est *statique*, pas *dynamique* (cf. `x2` et `x1`).
- En forme SSA, la définition *domine* les usages.

Forme SSA : staticité et dominance



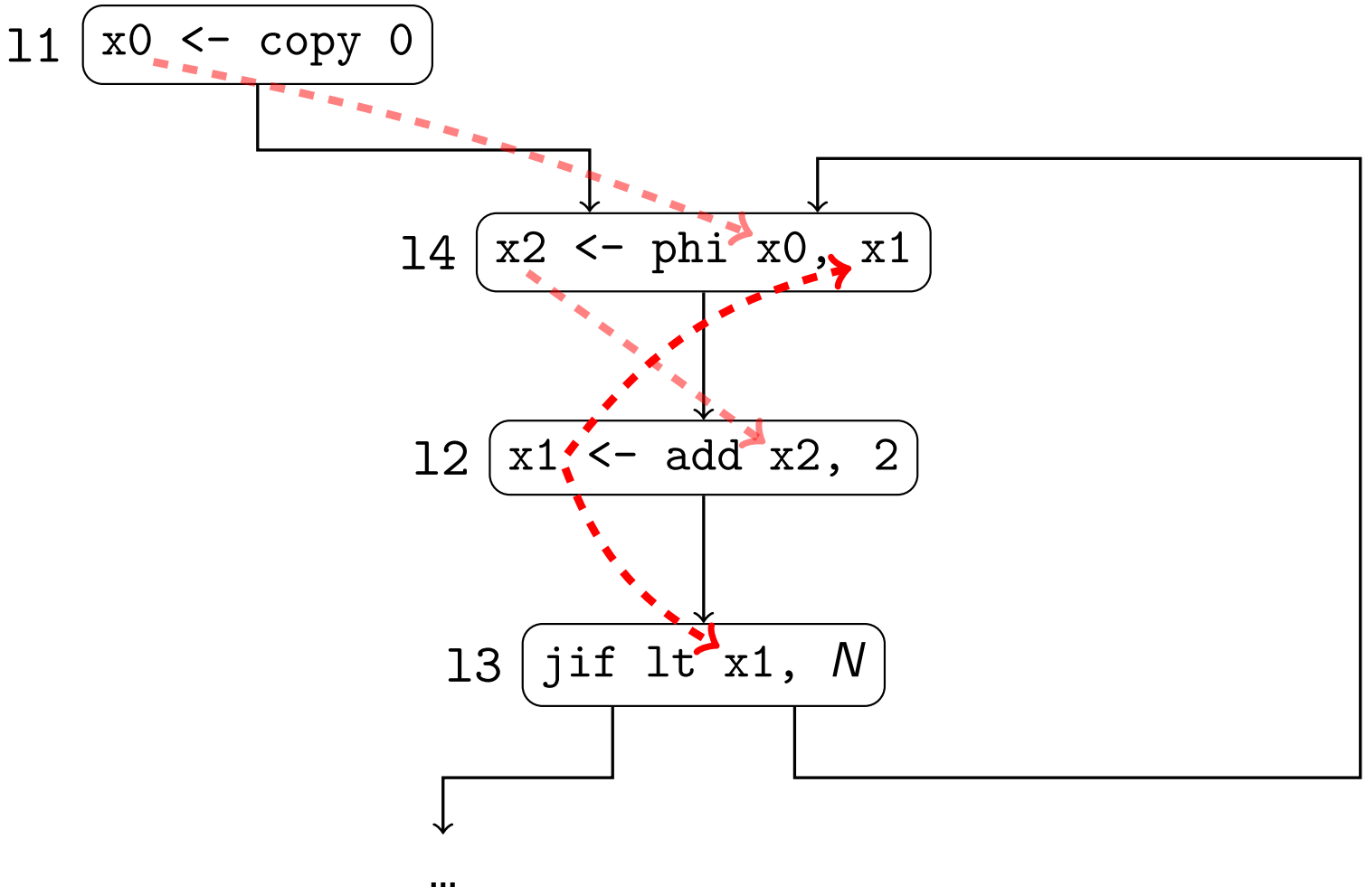
- L'unicité des définitions est *statique*, pas *dynamique* (cf. `x2` et `x1`).
- En forme SSA, la définition *domine* les usages.

Forme SSA : staticité et dominance



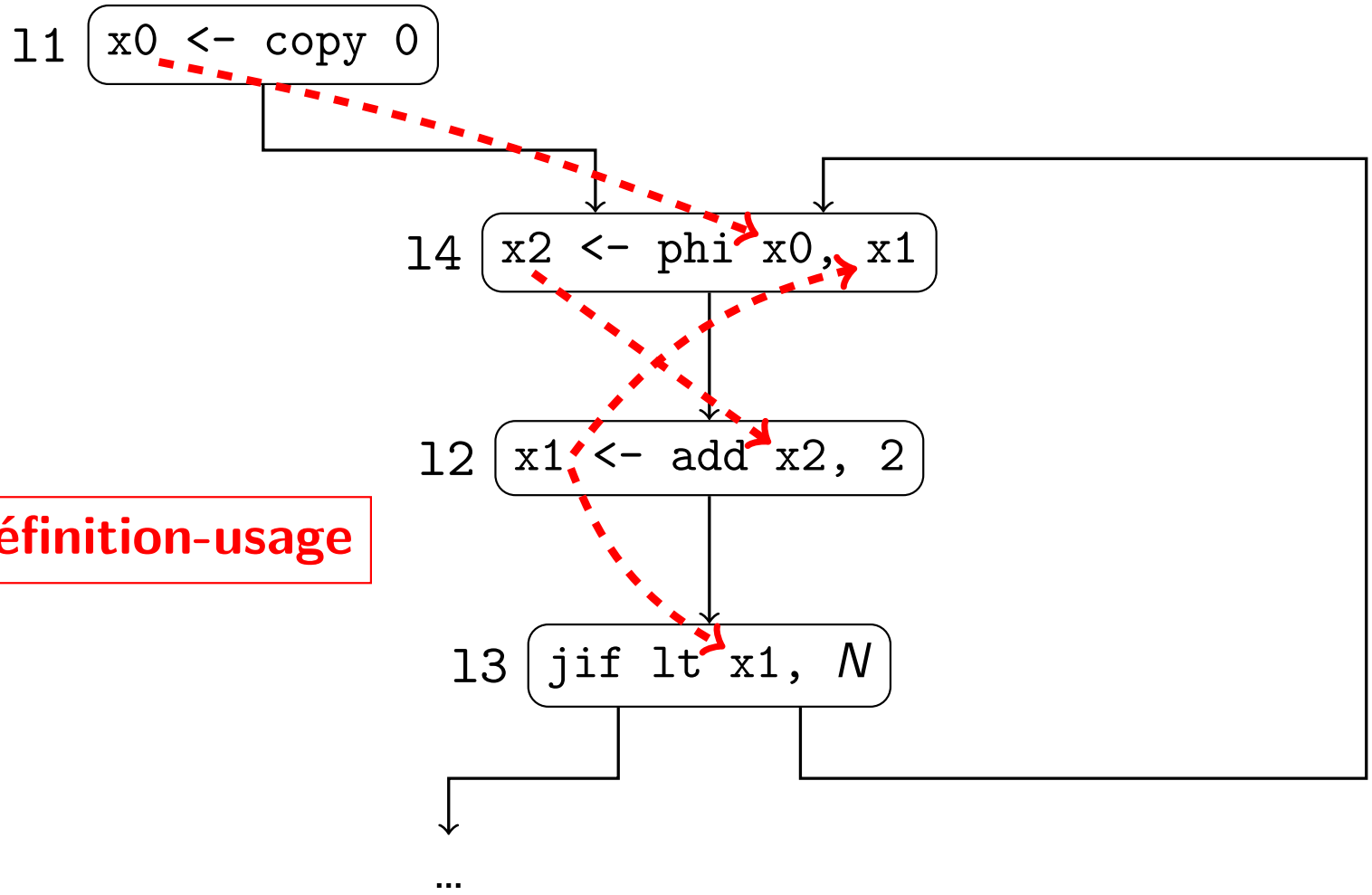
- L'unicité des définitions est *statique*, pas *dynamique* (cf. `x2` et `x1`).
- En forme SSA, la définition *domine* les usages.

Forme SSA : staticité et dominance



- L'unicité des définitions est *statique*, pas *dynamique* (cf. `x2` et `x1`).
- En forme SSA, la définition *domine* les usages.

Forme SSA : staticité et dominance



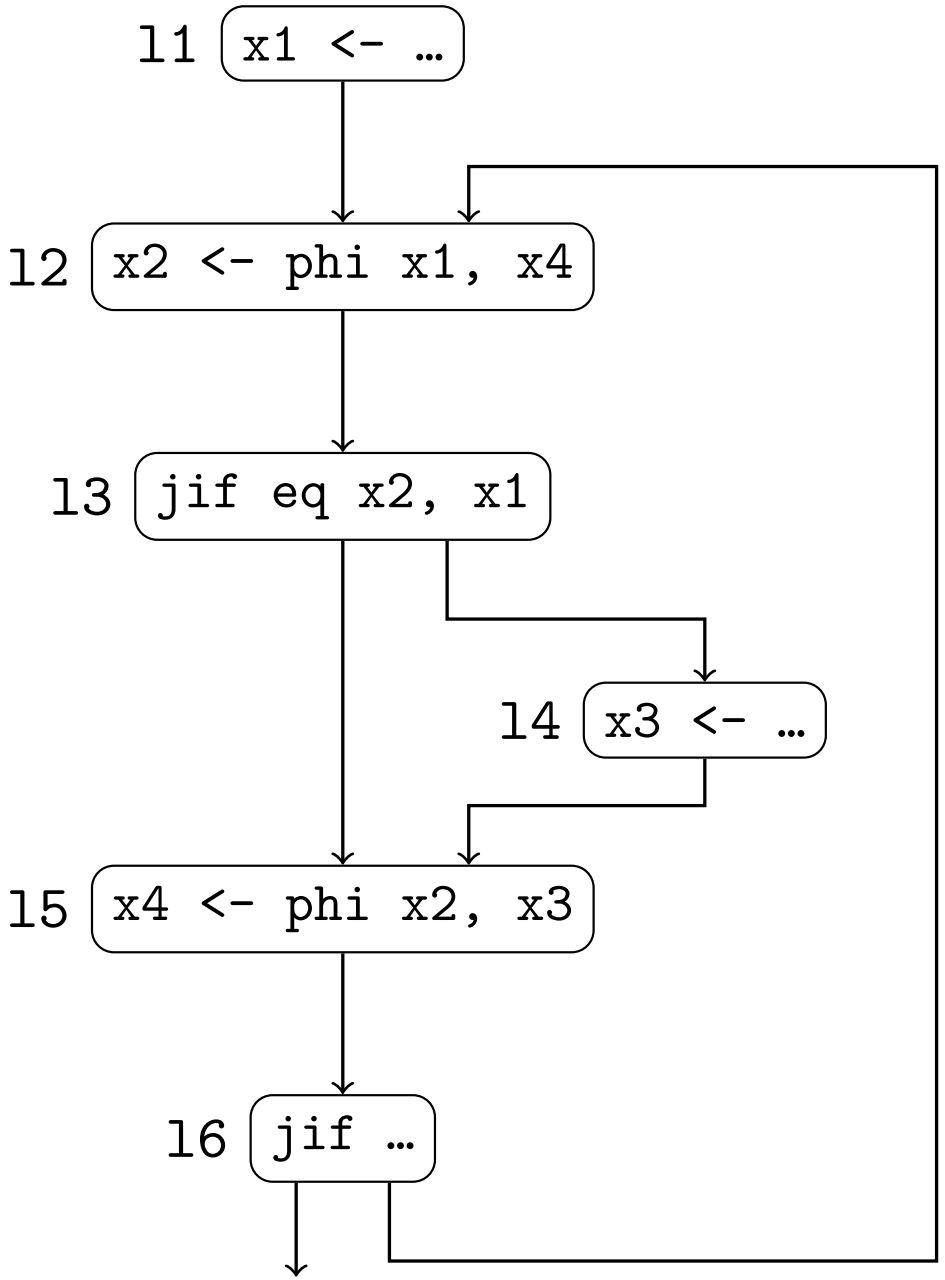
Chaînes définition-usage

- L'unicité des définitions est *statique*, pas *dynamique* (cf. `x2` et `x1`).
- En forme SSA, la définition *domine* les usages.

Application : propagation de copie parcimonieuse

(Analyse parcimonieuse en suivant les chaînes définition-usage : au tableau.)

x0	
x1	
x2	
x3	



La forme SSA aujourd'hui

La forme SSA aujourd'hui

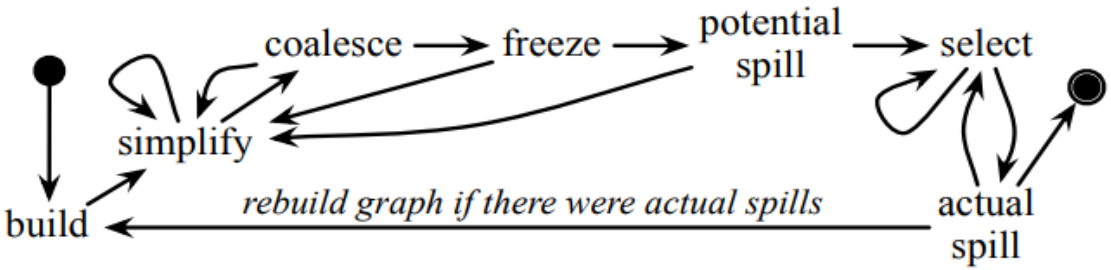
- Simplifie la conception d'analyses dataflow parcimonieuses.
  - Propriétés associées aux variables, suivi des chaînes *def-use*.
  - Socle de l'état de l'art en optimisation : SSAPRE, SCCP...

La forme SSA aujourd'hui

- Simplifie la conception d'analyses dataflow parcimonieuses.
  - Propriétés associées aux variables, suivi des chaînes *def-use*.
  - Socle de l'état de l'art en optimisation : SSAPRE, SCCP…
- Impacte très positivement l'allocation registre.
  - Graphes d'interférences *chordaux*, coloriables en temps polynomial.
  - Mène de l'allocation de registre itérative classique au vidage découplé.

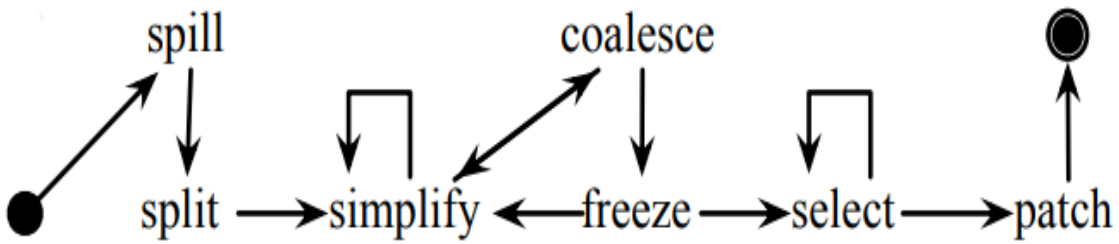
La forme SSA aujourd'hui

- Simplifie la conception d'analyses dataflow parcimonieuses.
  - Propriétés associées aux variables, suivi des chaînes *def-use*.
  - Socle de l'état de l'art en optimisation : SSAPRE, SCCP…
- Impacte très positivement l'allocation registre.
  - Graphes d'interférences *chordaux*, coloriables en temps polynomial.
  - Mène de l'allocation de registre itérative classique au vidage découplé.



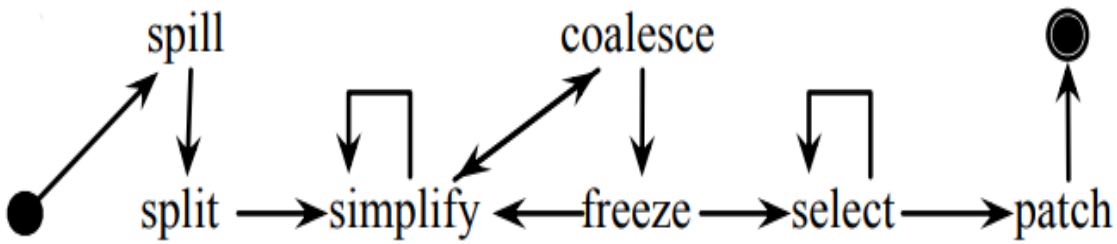
La forme SSA aujourd'hui

- Simplifie la conception d'analyses dataflow parcimonieuses.
  - Propriétés associées aux variables, suivi des chaînes *def-use*.
  - Socle de l'état de l'art en optimisation : SSAPRE, SCCP…
- Impacte très positivement l'allocation registre.
  - Graphes d'interférences *chordaux*, coloriables en temps polynomial.
  - Mène de l'allocation de registre itérative classique au vidage découplé.



La forme SSA aujourd'hui

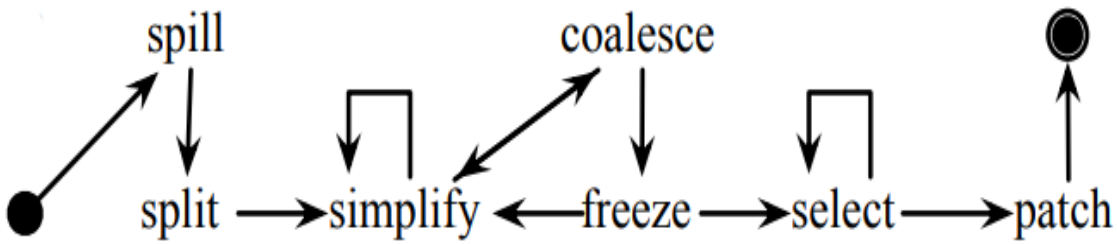
- Simplifie la conception d'analyses dataflow parcimonieuses.
  - Propriétés associées aux variables, suivi des chaînes *def-use*.
  - Socle de l'état de l'art en optimisation : SSAPRE, SCCP…
- Impacte très positivement l'allocation registre.
  - Graphes d'interférences *chordaux*, coloriables en temps polynomial.
  - Mène de l'allocation de registre itérative classique au vidage découplé.



- Quelques difficultés et points délicats, heureusement bien compris.
  - Construction : placement des fonctions ϕ en fonction de la dominance.
  - Destruction : élimination optimisée des fonctions ϕ.

La forme SSA aujourd'hui

- Simplifie la conception d'analyses dataflow parcimonieuses.
  - Propriétés associées aux variables, suivi des chaînes *def-use*.
  - Socle de l'état de l'art en optimisation : SSAPRE, SCCP…
- Impacte très positivement l'allocation registre.
  - Graphes d'interférences *chordaux*, coloriables en temps polynomial.
  - Mène de l'allocation de registre itérative classique au vidage découplé.



- Quelques difficultés et points délicats, heureusement bien compris.
  - Construction : placement des fonctions ϕ en fonction de la dominance.
  - Destruction : élimination optimisée des fonctions ϕ.
- Intégrée aux compilateurs industriels : GCC 4.0+, LLVM 1.0+.

Outline

Conclusion

Analyse dataflow

- Fournit une surapproximation du comportement du programme pour permettre certaines optimisations.
- Résultats obtenus par calcul de point fixe dans un ensemble ordonné.

Forme SSA

- Améliore la représentation des dépendances dans le code impératif.
- Guide la conception des compilateurs optimisants depuis fin 1980.

Références :

- *Modern Compiler Implementation* d'Appel.
  - Chapitres 10 et 17.
- *Principles of Static Analysis*, de Nielsen, Nielsen et Hankin.
  - Chapitre 2.
- Le *SSA Book* de Rastello et coauteurs.
  - Chapitres 1, 7 et 10.