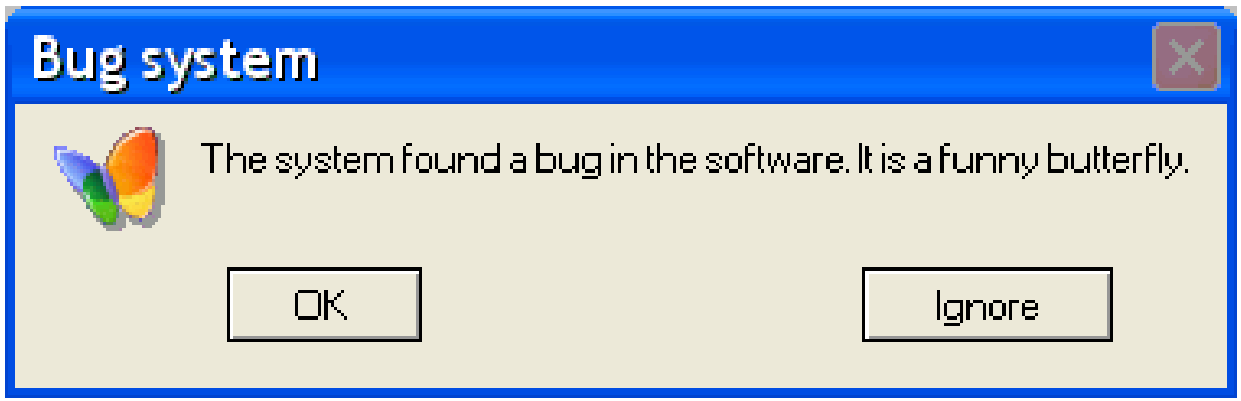
Un tour dans les étoiles

Étienne Lozes

ENS Cachan & RWTH Aachen

3 décembre 2009

La chasse au bug



Les bugs liés à l'allocation mémoire explicite:

- Violation mémoire
- Fuite mémoire
- Double déallocation

La jungle où on le trouve

- structures récursives “simples” (listes, arbres)
- structures récursives optimisées (listes doublement chaînes, listes rapides,...)
- programmation objets
- concurrence
- utilisé surtout en système (noyau, drivers,...)

```
1  void* SinglyLinkedPool::Acquire (size_t bytes)
2  {
3      unsigned int const blocks =
4          (bytes + sizeof (Header) + sizeof (Block) -
5              sizeof (Block);
6
7      Block* prevPtr = &sentinel;
8      Block* ptr = prevPtr->next;
9      while (ptr != 0 && ptr->length < blocks)
10     {
11         prevPtr = ptr;
12         ptr = ptr->next;
13     }
14     if (ptr == 0)
15         throw bad_alloc ("out of memory");
16     if (ptr->length > blocks)
17     {
18         Block& newBlock = ptr [blocks];
19         newBlock.length = ptr->length - blocks;
20         newBlock.next = ptr->next;
21         ptr->length = blocks;
22         ptr->next = &newBlock;
23     }
24     prevPtr->next = ptr->next;
25     return ptr->userPart;
26 }
```

Un inventaire (sans doute très incomplet)

Sémantique
BI,
SL,
Boolean BI vs BI,
HO SL,
anti-frame,
Classical BI,
Locks in heap,
Abstract SL,
Footprint.

Concurrence
Concurrent
Separation Logic,
Permissions,
Message passing,
Rely Guarantee SL,
Deny Guarantee.

Automatisation
Smallfoot
Th. de Brochenin
Space Invader,
jStar,
Bi-abduction,
SmallfootRG,
HeapHop

Un inventaire (sans doute très incomplet)

Sémantique

BI,
SL,
Boolean BI vs BI,
HO SL,
anti-frame,
Classical BI,
Locks in heap,
Abstract SL,
Footprint.

Concurrence

Concurrent
Separation Logic,
Permissions,
Message passing,
Rely Guarantee SL,
Deny Guarantee.

Automatisation

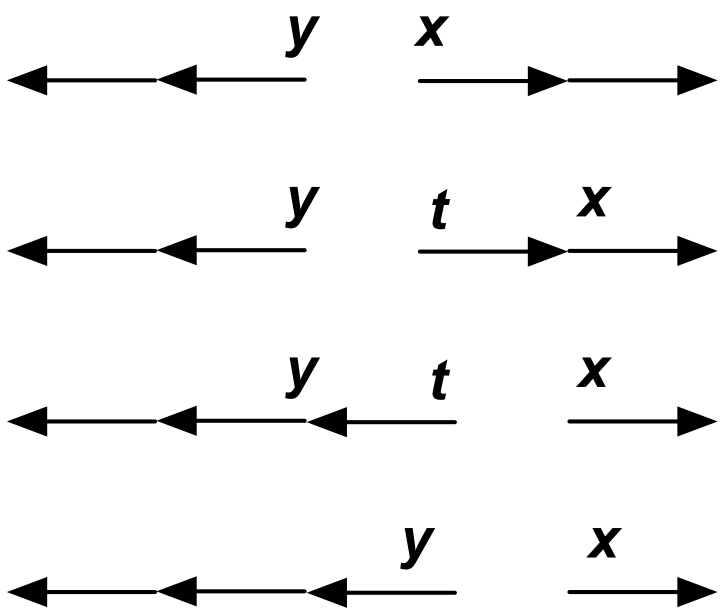
Smallfoot
Th. de Brochenin
Space Invader,
jStar,
Bi-abduction,
SmallfootRG,
HeapHop

Plan

Les têtes à poux

Reverse and append lists

```
function
revappend(x,y)
while x<>null do
  t:=x;
  x=[t];
  [t]=y;
  y:=t;
end
```

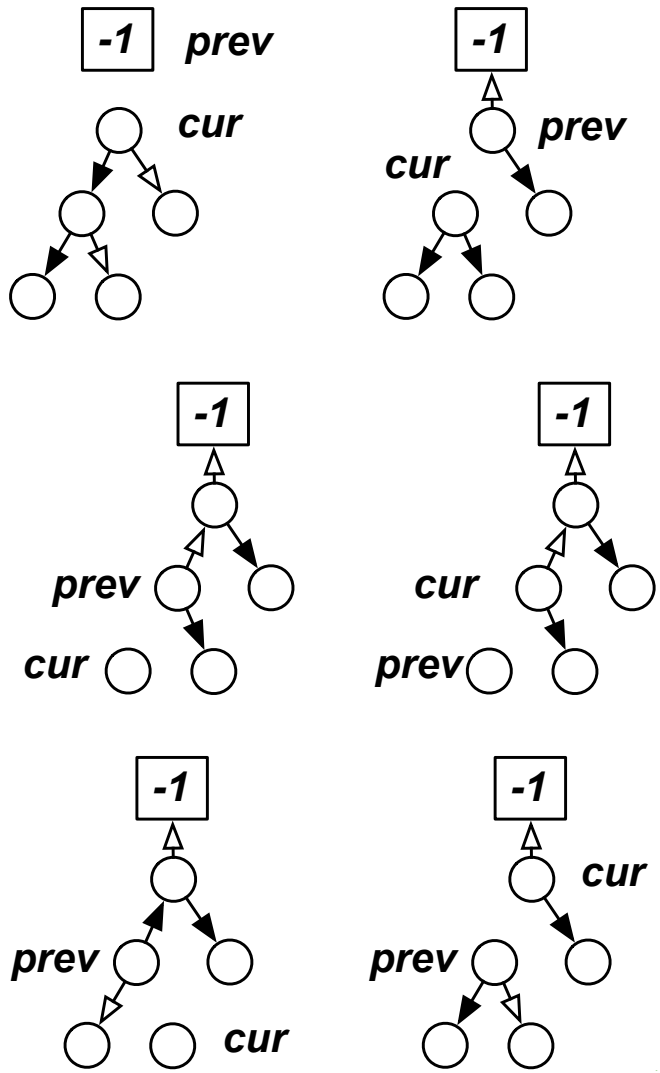


Question

Quelle est la condition nécessaire et suffisante sur la mémoire initiale pour que ce programme termine (sans erreur)? diverge?

L'algorithme Deutsch Schorr Waite

```
if (root == NULL) return;
prev = -1; cur = root;
while (true) {
  next = cur->left;
  cur->left = cur->right;
  cur->right = prev;
  prev = cur;
  cur = next;
  if (cur == -1) break;
  if (cur == NULL) {
    cur = prev;
    prev = NULL;
  }
}
```



Un schéma classique: lecteurs et rédacteurs

LECTEURS

```
with read do
  if count = 0 lock(write);
  count++;
done
```

```
...:= a->f
```

```
with read when count>0 do
  count--;
  if (count = 0) unlock(write);
done
```

REDACTEURS

```
lock(write);
```

```
a->f :=...
```

```
unlock(write)
```

Une pile de Treiber

```
atomic bool CAS(a,b,c){
if (*a==b) {*a=c;return true;} return false;}
```

Une pile de Treiber

```
atomic bool CAS(a,b,c){
if (*a==b) {*a=c;return true;} return false;}
```

```
void push(value  v) {
  cell *t, *x;
  x = alloc();
  x->data = v;
  do {
    t = top_ptr;
    x->next = t;
  } while (!CAS(&top_ptr,t,x));
}
```

```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next;
  }
  while (!CAS(&top_ptr,t,x));
  return t->data;
}
```

La pile de Treiber

- «ça marche »
- pas de verrou, donc pas de deadlock possible
- pas de “livelock” (voir conditions de progrès)
- il existe plus compliqué, cf livre de Herlihy.

La même pile sans GC

```
void push(value  v) {
  cell *t, *x;
  x = alloc();
  x->data = v;
  do {
    t = top_ptr;
    x->next = t;
  } while (!CAS(&top_ptr,t,x));
}
```

```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next;
  }
  while (!CAS(&top_ptr,t,x));
  data_t data = t->data;
  free(t);
  return data;
}
```

La même pile sans GC

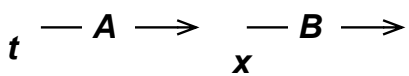
```
void push(value  v) {
  cell *t, *x;
  x = alloc();
  x->data = v;
  do {
    t = top_ptr;
    x->next = t;
  } while (!CAS(&top_ptr,t,x));
}
```

```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next;
  }
  while (!CAS(&top_ptr,t,x));
  data_t data = t->data;
  free(t);
  return data;
}
```

«Ca ne marche pas!!»

Le problème ABA, importance du GC

1. on part de la pile `A::B::nil`.
2. Thread 1 commence pop et est préempté



5.
```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next; (1 STOP)
  }
  while (!CAS(&top_ptr,t,x));
  data_t data = t->data;
  free(t);
  return data;
}
```

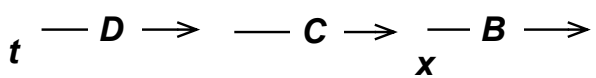
Le problème ABA, importance du GC

1. on part de la pile `A::B::nil`.
2. Thread 1 commence pop et est préempté
3. Thread 2 fait `pop:A` (t de 1 est maintenant libre)

5.
```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next; (1 STOP)
  }
  while (!CAS(&top_ptr,t,x));
  data_t data = t->data;
  free(t);
  return data;
}
```

Le problème ABA, importance du GC

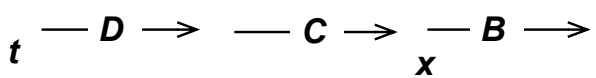
1. on part de la pile `A::B::nil`.
2. Thread 1 commence pop et est préempté
3. Thread 2 fait `pop:A`
4. Thread 2 fait `push(C);push(D)` et alloue D en t de 1



5.
```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next; (1 STOP)
  }
  while (!CAS(&top_ptr,t,x));
  data_t data = t->data;
  free(t);
  return data;
}
```

Le problème ABA, importance du GC

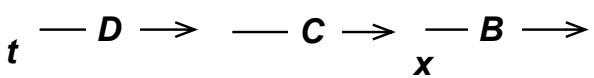
1. on part de la pile `A::B::nil`.
2. Thread 1 commence pop et est préempté
3. Thread 2 fait `pop:A`
4. Thread 2 fait `push(C);push(D)` et alloue D en t de 1
5. Thread 1 reprend et passe le test




```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next; (1 STOP)
  }
  while (!CAS(&top_ptr,t,x));
  data_t data = t->data;
  free(t);
  return data;
}
```

Le problème ABA, importance du GC

1. on part de la pile `A::B::nil`.
2. Thread 1 commence pop et est préempté
3. Thread 2 fait `pop:A`
4. Thread 2 fait `push(C);push(D)` et alloue D en t de 1
5. Thread 1 supprime C,D d'un coup `pop:D`, puis pop encore `pop:B`




```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next; (1 STOP)
  }
  while (!CAS(&top_ptr,t,x));
  data_t data = t->data;
  free(t);
  return data;
}
```

Le problème ABA, importance du GC

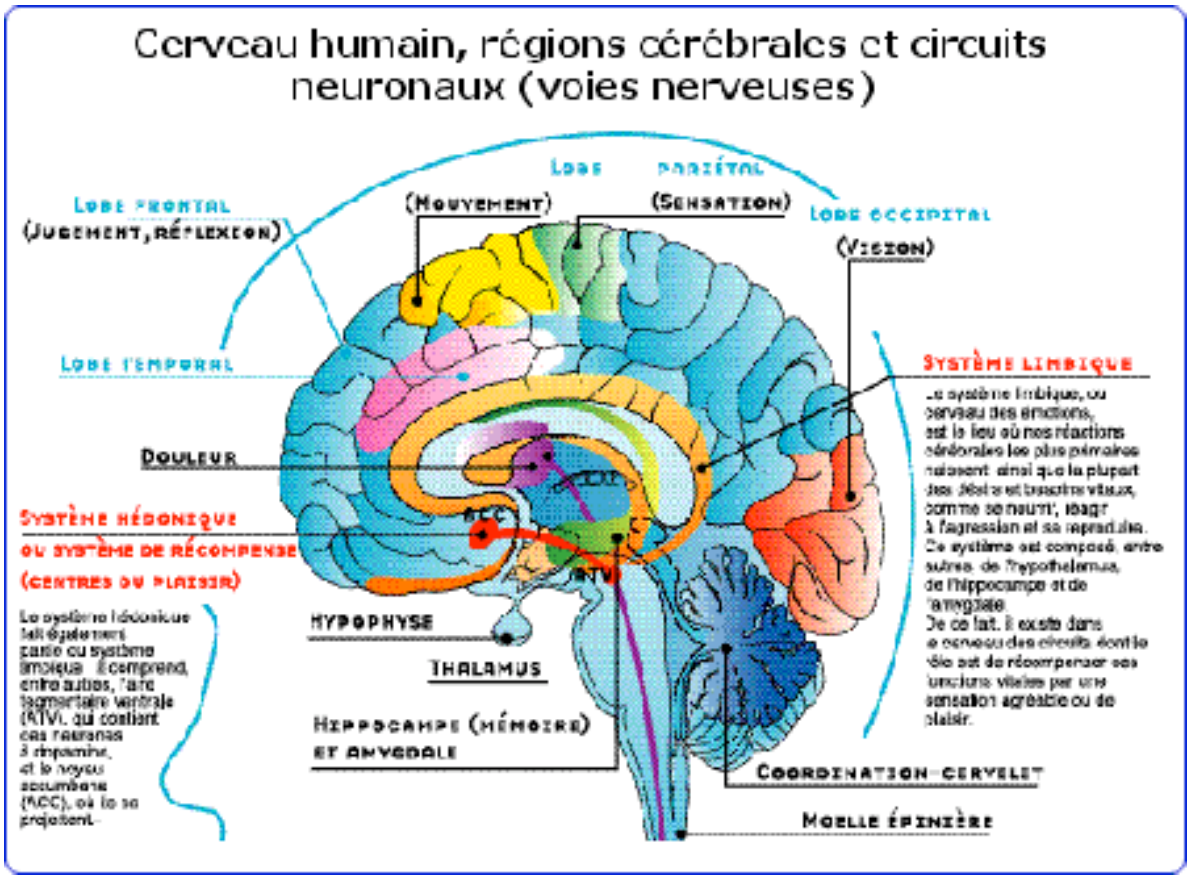
1. on part de la pile `A::B::nil`.
2. Thread 1 commence pop et est préempté
3. Thread 2 fait `pop:A`
4. Thread 2 fait `push(C);push(D)` et alloue D en t de 1
5. Thread 1 supprime C,D d'un coup `pop:D`, puis pop encore `pop:B`

```
value pop() {
  cell *t, *x;
  do {
    t = top_ptr;
    if (t == NULL) return EMPTY
    x = t->next; (1 STOP)
  }
  while (!CAS(&top_ptr,t,x));
  data_t data = t->data;
  free(t);
  return data;
}
```

`(pop:A;push(C);push(D)) || (pop:D;pop:B)`

Ceci n'est pas correct!

Modéliser la mémoire .

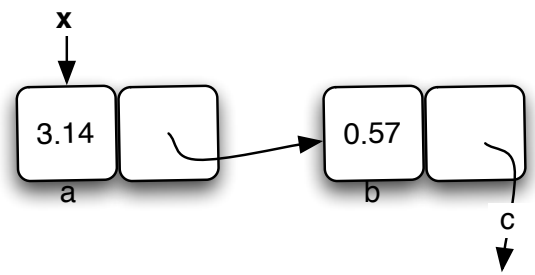


Un modèle générique Un état mémoire est un couple comportant

- une pile $s : \mathrm{Var} \rightharpoonup \mathrm{Val}$, dom s = variables déclarées
- un tas $h : \mathbb{N} \rightharpoonup \mathrm{Val}$, dom h = espace alloué.
- $\mathrm{Val} = \mathbb{N} \sqcup \mathrm{Dat} \sqcup \mathrm{Lk} \sqcup \ldots$
- sélecteur de champs = offset fixé

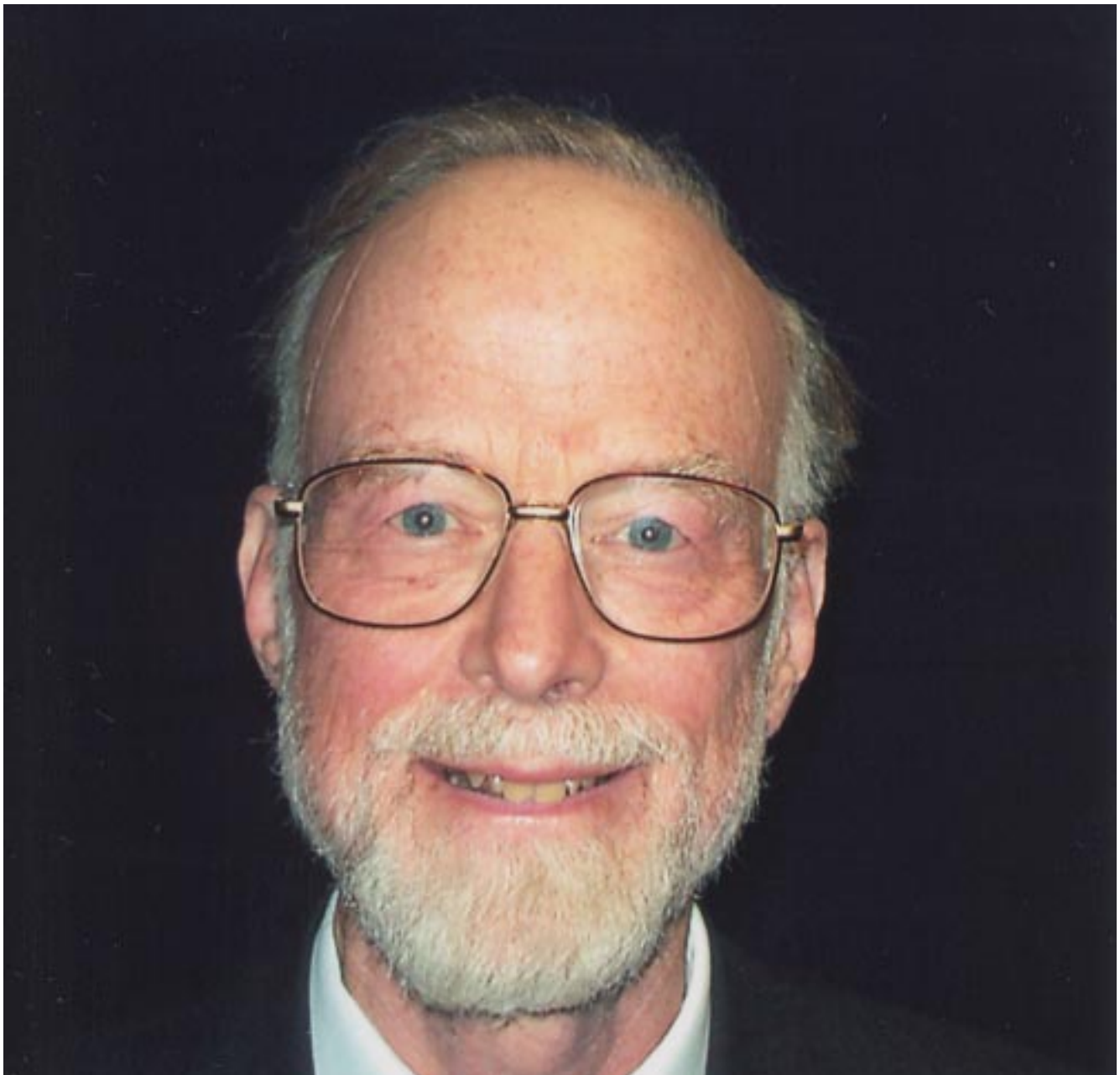
Exemple

$\mathrm{Var} = \{\mathbf{x}\}$
$s = \{\mathbf{x} \mapsto a\}$



$h = \{a \mapsto 3.14, a+1 \mapsto b, b \mapsto 0.57, b+1 \mapsto c)\}$

Le premier astronaute



La preuve de programmes selon Hoare/Floyd

1. On annote le programme par des *triplets* $\{\mathcal{A}\}\ \ P\ \ \{\mathcal{B}\}$:

Ex:

$$\{\top\}\ \mathbf{x} := 2\ \{\mathbf{x} = 2\}\ \mathbf{y} := 3\ \{\mathbf{y} = \mathbf{x} + 1\}$$

2. On fait une preuve à partir de règles élémentaires.

Ex:

$$\frac{\{\mathcal{A}\}\ P\ \{\mathcal{B}\} \quad \{\mathcal{B}\}\ Q\ \{\mathcal{C}\}}{\{\mathcal{A}\}\ P; Q\ \{\mathcal{C}\}} \qquad \frac{}{\{\mathcal{A}[e/x]\}\ x := e\ \{\mathcal{A}\}}$$

Un système de preuve

Backward axiom

$$\frac{}{\{\mathcal{A}[e/x]\}\ x := e\ \{\mathcal{A}\}}$$

Invariant rule

$$\frac{\{\mathcal{I} \wedge b\}\ P\ \{\mathcal{I}\}}{\{\mathcal{I}\}\ \textbf{while}\ b\ \textbf{do}\ P\ \{\mathcal{I} \wedge \neg b\}}$$

Cut rule

$$\frac{\{\mathcal{A}\}\ P\ \{\mathcal{B}\} \quad \{\mathcal{B}\}\ Q\ \{\mathcal{C}\}}{\{\mathcal{A}\}\ P; Q\ \{\mathcal{C}\}}$$

Logical rule

$$\frac{\mathcal{A} \Rightarrow \mathcal{A}' \quad \{\mathcal{A}'\}\ P\ \{\mathcal{B}'\} \quad \mathcal{B}' \Rightarrow \mathcal{B}}{\{\mathcal{A}\}\ P\ \{\mathcal{B}\}}$$

Propriété de correction

Si il existe une preuve de $\{\mathcal{A}\}\ P\ \{\mathcal{B}\}$, alors le triplet est valide: $[\![P]\!]([\![\mathcal{A}]\!]) \subseteq [\![\mathcal{B}]\!])$.

Propriété de complétude

Est-ce que tous les triplets valides sont prouvables?

Precondition la plus faible

Plus faible précondition

Etant donnés $P, \mathcal{B}$, la plus faible précondition $w(P, \mathcal{B})$ est telle que pout toute condition $\mathcal{A}$ impliquant $\{\mathcal{A}\}\ P\ \{\mathcal{B}\}$, on a $\mathcal{A} \Rightarrow w(P, \mathcal{B})$.

Théorème [Scott]

Si pour tout programme P et tout $\mathcal{B}$, $w(P, \mathcal{B})$ s'exprime, alors le système est complet.

$\Rightarrow$ Dépend du langage logique!

Corrolaire

Si on a les préconditions des instructions atomiques et le second ordre, le système est complet.

Difficultés avec les pointeurs

$s, h \models \mathbf{x} \hookrightarrow e, e'$ ssi $h \circ s(\mathbf{x}) = (\llbracket e \rrbracket, \llbracket e' \rrbracket)$.

Problèmes d'aliasing

- $\{\mathbf{x} \hookrightarrow - \wedge \mathbf{y} \hookrightarrow -\}$ dispose $\mathbf{x}$ $\{\mathbf{y} \hookrightarrow -\}$
- $\dfrac{\{\mathcal{A}\}\ P\ \{\mathcal{B}\}}{\{\mathcal{A} \wedge \mathcal{H}\}\ P\ \{\mathcal{B} \wedge \mathcal{H}\}}$ avec $\mathrm{fv}(\mathcal{H}) \cap \mathrm{fv}(P, \mathcal{A}, \mathcal{B}) = \emptyset$

Calcul de plus faible précondition

- pas évident le sens de : $\{\mathcal{A}[\mathbf{x} \to f := y]\}$ $\mathbf{x} \to f := y$ $\{\mathcal{A}\}$
- pire : $\{???\}$ dispose $\mathbf{x}$ $\{\mathcal{A}\}$

Le traitement des erreurs

Une subtilité du triplet de Hoare

$\{\top\}\ P\ \{\bot\}$ est valide si P «ne termine pas»... mais ne fait pas d'erreur.

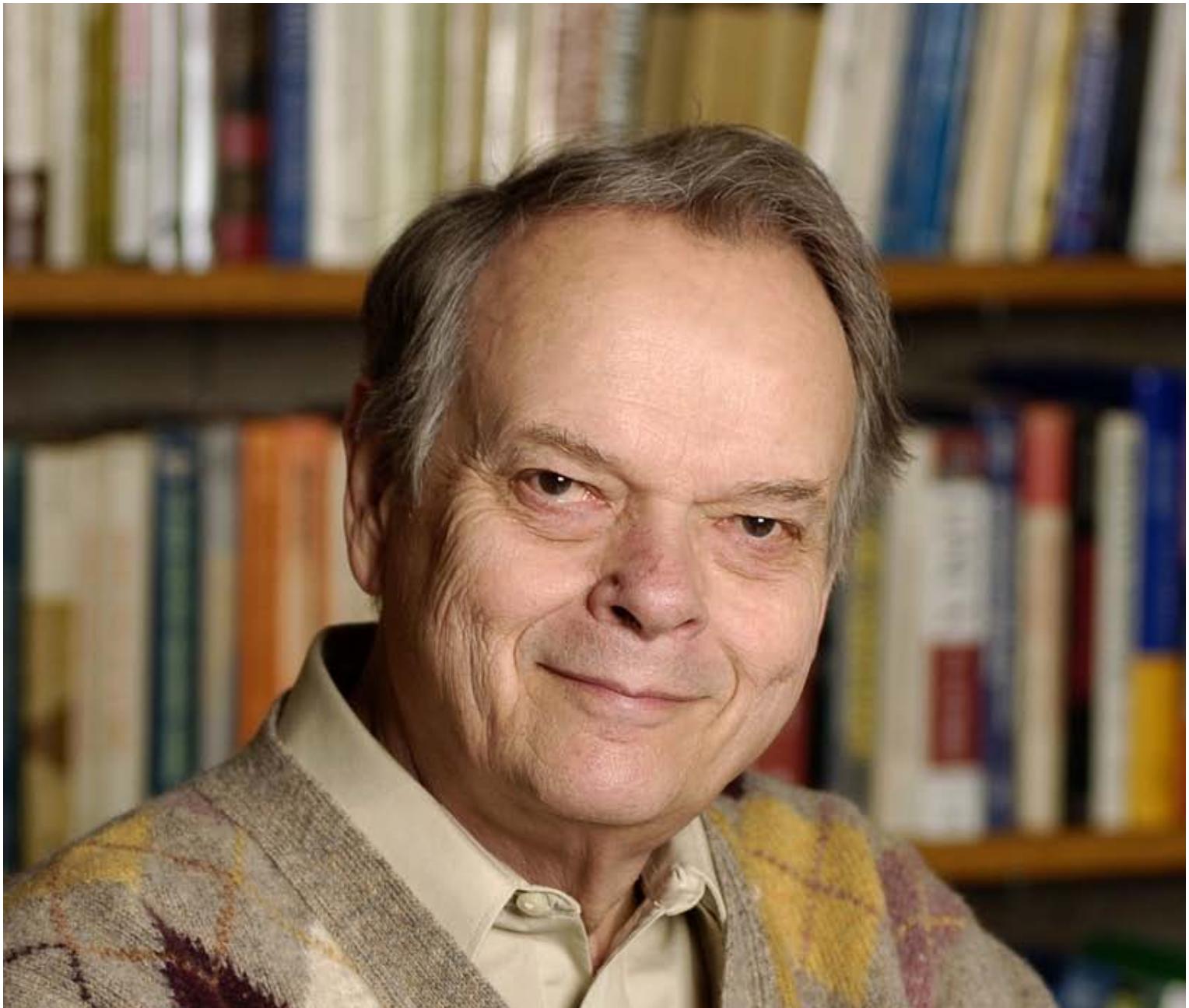
Les non-terminaisons et les erreurs

- divergence (non-terminaison)
- interblocage (non-terminaison)
- violation mémoire (erreur)
- double déallocation (erreur)

Et les fuites mémoires?

- pas nécessairement une erreur
- on admet en général que $\{\mathsf{emp}\}\ P\ \{\mathsf{emp}\}$ signifie pas de fuite mémoire, mais subtil avec la concurrence, en particulier échange de messages.

Le deuxième astronaute



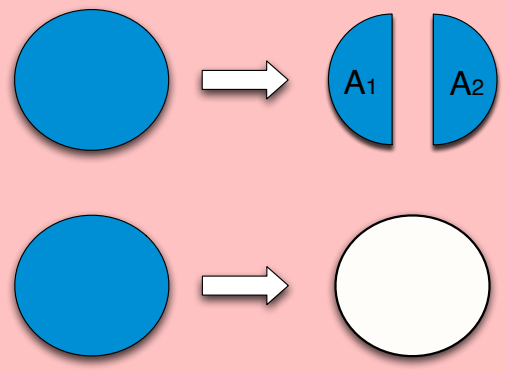
Raisonnement **local**: la conjonction séparante

Union disjointe de mémoire:

$$(s,h) = (s_1,h_1) \bullet (s_2,h_2) \quad \text{si} \quad \begin{cases} h = h_1 \cup h_2, s = s_1 \cup s_2 \\ dom\ s_1 \cap dom\ s_2 = \emptyset \\ dom\ h_1 \cap dom\ h_2 = \emptyset \end{cases}$$

La conjonction séparante

- $s,h \models \mathcal{A}_1 * \mathcal{A}_2$ ssi $\exists s_1,h_1,s_2 h_2.$ $(s,h) = (s_1,h_1) \bullet (s_2,h_2)$, $s_1,h_1 \models \mathcal{A}_1$ et $s_2,h_2 \models \mathcal{A}_2$
- $s,h \models \text{emp}$ ssi $h = \emptyset$



Quelques (non) identités:

$$\mathcal{A} * \mathcal{B} = \mathcal{B} * \mathcal{A} \qquad \mathcal{A} * \bot = \bot \qquad \mathcal{A} * \text{emp} = \mathcal{A}$$
$$\mathcal{A} * \mathcal{A} \neq \mathcal{A} \qquad \mathcal{A} * \top \neq \mathcal{A} \qquad \mathcal{A} * \neg\mathcal{A} \neq \bot$$

Plus de problèmes d'aliasing

- $\{(\mathbf{x} \hookrightarrow -) * (\mathbf{y} \hookrightarrow -)\}$ dispose $\mathbf{x}$ $\{\mathbf{y} \hookrightarrow -\}$
- $$\frac{\{\mathcal{A}\}\ P\ \{\mathcal{B}\}}{\{\mathcal{A} * \mathcal{H}\}\ P\ \{\mathcal{B} * \mathcal{H}\}}$$

Exemples d'utilisation

Plus de problèmes d'aliasing

- $\{(\mathbf{x} \hookrightarrow -) * (\mathbf{y} \hookrightarrow -)\}$ dispose $\mathbf{x}$ $\{\mathbf{y} \hookrightarrow -\}$
- $$\frac{\{\mathcal{A}\}\ P\ \{\mathcal{B}\}}{\{\mathcal{A} * \mathcal{H}\}\ P\ \{\mathcal{B} * \mathcal{H}\}}$$

Evaluer la mémoire allouée

$$\underbrace{\neg\mathsf{emp} * \neg\mathsf{emp} * \ldots * \neg\mathsf{emp}}_{k \text{ fois}} == \sharp dom h \geq k$$

Formule "précise"

$e \mapsto f : a \triangleq e \hookrightarrow f : a \wedge \neg(\neg\mathsf{emp} * \neg\mathsf{emp})$

$dom\ h = \{[\![e]\!]\}$

Le noyau dur de la logique de séparation

Frame rule

$$\frac{\{\mathcal{A}\}\ P\ \{\mathcal{B}\}}{\{\mathcal{A} * \mathcal{F}\}\ P\ \{\mathcal{B} * \mathcal{F}\}}$$

Disjoint Concurrency rule

$$\frac{\{\mathcal{A}_1\}\ P_1\ \{\mathcal{B}_1\} \quad \{\mathcal{A}_2\}\ P_2\ \{\mathcal{B}_2\}}{\{\mathcal{A}_1 * \mathcal{A}_2\}\ P_1 || P_2\ \{\mathcal{B}_1 * \mathcal{B}_2\}}$$

Small axioms

$\{\mathsf{emp}\}$	$\mathbf{x} = \mathrm{new}()$	$\{\mathbf{x} \mapsto -\}$
$\{\mathbf{x} \mapsto -\}$	$\mathrm{dispose}\ \mathbf{x}$	$\{\mathsf{emp}\}$
$\{\mathbf{x} \mapsto f : -\}$	$\mathbf{x} \to f := \mathbf{y}$	$\{\mathbf{x} \mapsto f : \mathbf{y}\}$
$\{\mathbf{y} \mapsto f : -\}$	$\mathbf{x} := \mathbf{y} \to f$	$\{\mathbf{y} \mapsto f : \mathbf{x}\}$

Fonctions

$$\frac{\{\mathcal{A}\}\ f(\vec{x})\ \{\mathcal{B}\} \quad \in \quad \Gamma}{\Gamma \ \vdash \ \{\mathcal{A}[\vec{x} := \vec{E}]\}\ f(\vec{E})\ \{\mathcal{B}[\vec{x} := \vec{E}]\}}$$

Exemples

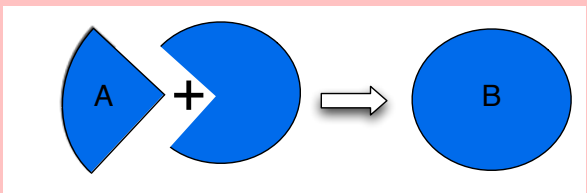
- list.sf
- tree.sf

Raisonnement **contextuel**: l'implication spatiale

Un fondement théorique:
La « Bunched Logic »:

L'implication spatiale (aka magic wand, adjunct,...)

$s, h \models \mathcal{A} \mathrel{-\!\!*} \mathcal{B}$ ssi
$\forall s', h'$, si $(s, h) \bullet (s', h')$ est défini
et si $s', h' \models \mathcal{A}$,
alors $(s, h) \bullet (s' h') \models \mathcal{B}$



Quelques identités:

$$
\begin{array}{lcl}
\mathcal{A} * (\mathcal{A} \mathrel{-\!\!*} \mathcal{B}) & \Rightarrow & \mathcal{B} \\
\mathsf{emp} \mathrel{-\!\!*} \mathcal{A} & \Leftrightarrow & \mathcal{A} \\
\mathcal{A} \mathrel{-\!\!*} (\mathcal{B} \mathrel{-\!\!*} \mathcal{C}) & \Leftrightarrow & (\mathcal{A} * \mathcal{B}) \mathrel{-\!\!*} \mathcal{C} \\
(\mathbf{x} \mapsto -) \mathrel{-\!\!*} \bot & \Leftrightarrow & (\mathbf{x} \mapsto -) * \top \\
\mathcal{A} \vdash \mathcal{B} \mathrel{-\!\!*} \mathcal{C} & \text{ssi} & \mathcal{A} * \mathcal{B} \vdash \mathcal{C}
\end{array}
$$

Raisonnement arrière en logique de séparation

Notation: $WP(P, \mathcal{A})$ = plus faible précondition
i.e. : si $\{\mathcal{B}\}\ P\ \{\mathcal{A}\}$, alors $\mathcal{B} \Rightarrow w(P, \mathcal{A})$.

Un calcul de plus faibles préconditions

$WP(\text{dispose } \mathbf{x}\ ,\ \mathcal{A}) \quad =$

$WP(\mathbf{x} := \text{new}()\ ,\ \mathcal{A}) \quad =$

$WP(x \to f := \mathbf{y}\ ,\ \mathcal{A}) \quad =$

$WP(\mathbf{x} := \mathbf{y} \to f\ ,\ \mathcal{A}) \quad =$

Raisonnement arrière en logique de séparation

Notation: $WP(P, \mathcal{A})$ = plus faible précondition
i.e. : si $\{\mathcal{B}\}\ P\ \{\mathcal{A}\}$, alors $\mathcal{B} \Rightarrow w(P, \mathcal{A})$.

Un calcul de plus faibles préconditions

$$WP(\text{dispose } \mathbf{x}\ ,\ \mathcal{A}) \quad = \quad (\mathbf{x} \mapsto -) \ * \ \mathcal{A}$$

$$WP(\mathbf{x} := \text{new}()\ ,\ \mathcal{A}) \quad =$$

$$WP(x \to f := \mathbf{y}\ ,\ \mathcal{A}) \quad =$$

$$WP(\mathbf{x} := \mathbf{y} \to f\ ,\ \mathcal{A}) \quad =$$

Raisonnement arrière en logique de séparation

Notation: $WP(P, \mathcal{A})$ = plus faible précondition
i.e. : si $\{\mathcal{B}\}\ P\ \{\mathcal{A}\}$, alors $\mathcal{B} \Rightarrow w(P, \mathcal{A})$.

Un calcul de plus faibles préconditions

$$WP(\text{dispose } \mathbf{x}\ ,\ \mathcal{A}) \quad = \quad (\mathbf{x} \mapsto -) \ * \ \mathcal{A}$$

$$WP(\mathbf{x} := \text{new}()\ ,\ \mathcal{A}) \quad = \quad (\mathbf{x} \mapsto -) \ -\!\!* \ \mathcal{A}$$

$$WP(x \to f := \mathbf{y}\ ,\ \mathcal{A}) \quad =$$

$$WP(\mathbf{x} := \mathbf{y} \to f\ ,\ \mathcal{A}) \quad =$$

Raisonnement arrière en logique de séparation

Notation: $WP(P, \mathcal{A})$ = plus faible précondition
i.e. : si $\{\mathcal{B}\}\ P\ \{\mathcal{A}\}$, alors $\mathcal{B} \Rightarrow w(P, \mathcal{A})$.

Un calcul de plus faibles préconditions

$$WP(\text{dispose } \mathbf{x}\ ,\ \mathcal{A}) \quad = \quad (\mathbf{x} \mapsto -) \ * \ \mathcal{A}$$

$$WP(\mathbf{x} := \text{new}()\ ,\ \mathcal{A}) \quad = \quad \forall x'.\ (x' \mapsto -) \ \mathrel{-\!\!*}\ \mathcal{A}'$$

$$WP(x \to f := \mathbf{y}\ ,\ \mathcal{A}) \quad =$$

$$WP(\mathbf{x} := \mathbf{y} \to f\ ,\ \mathcal{A}) \quad =$$

Raisonnement arrière en logique de séparation

Notation: $WP(P, \mathcal{A})$ = plus faible précondition
i.e. : si $\{\mathcal{B}\}\ P\ \{\mathcal{A}\}$, alors $\mathcal{B} \Rightarrow w(P, \mathcal{A})$.

Un calcul de plus faibles préconditions

$$WP(\text{dispose } \mathbf{x}\ ,\ \mathcal{A}) \quad = \quad (\mathbf{x} \mapsto -) \ * \ \mathcal{A}$$

$$WP(\mathbf{x} := \text{new}()\ ,\ \mathcal{A}) \quad = \quad \forall x'.\ (x' \mapsto -) \ \mathrel{-\!\!*} \ \mathcal{A}'$$

$$WP(x \to f := \mathbf{y}\ ,\ \mathcal{A}) \quad = \quad (\mathbf{x} \mapsto f : -) \ * \ ((\mathbf{x} \mapsto f : \mathbf{y}) \mathrel{-\!\!*} \mathcal{A})$$

$$WP(\mathbf{x} := \mathbf{y} \to f\ ,\ \mathcal{A}) \quad =$$

Raisonnement arrière en logique de séparation

Notation: $WP(P, \mathcal{A})$ = plus faible précondition
i.e. : si $\{\mathcal{B}\}\ P\ \{\mathcal{A}\}$, alors $\mathcal{B} \Rightarrow w(P, \mathcal{A})$.

Un calcul de plus faibles préconditions

$$WP(\text{dispose } \mathbf{x}\ ,\ \mathcal{A}) \quad = \quad (\mathbf{x} \mapsto -) \ * \ \mathcal{A}$$

$$WP(\mathbf{x} := \text{new}()\ ,\ \mathcal{A}) \quad = \quad \forall x'.\ (x' \mapsto -) \ \mathrel{-\!\!*} \ \mathcal{A}'$$

$$WP(x \to f := \mathbf{y}\ ,\ \mathcal{A}) \quad = \quad (\mathbf{x} \mapsto f : -) \ * \ ((\mathbf{x} \mapsto f : \mathbf{y}) \mathrel{-\!\!*} \mathcal{A})$$

$$WP(\mathbf{x} := \mathbf{y} \to f\ ,\ \mathcal{A}) \quad = \quad \exists x'.\ (\mathbf{y} \hookrightarrow f : x') \ \wedge \ \mathcal{A}'$$

L'automatisation

Quelques problèmes de décision

Problème de vérification de preuve

Etant donnée une preuve de $\{\mathcal{A}\}\ P\ \{\mathcal{B}\}$, vérifier qu'elle est bien formée

Problème d'inférence de preuve faible

Etant donnés $\{\mathcal{A}\}\ P\ \{\mathcal{B}\}$, chercher une preuve.

Problème d'inférence de preuve forte

Etant donné P, chercher $\mathcal{A}, \mathcal{B}$ tels que $\{\mathcal{A}\}\ P\ \{\mathcal{B}\}$.

Impossible...

Tous ces problèmes sont indécidables en général:
$\{\top\}\ P\ \{\bot\}$ est valide ssi P diverge.

Les moyens d'y arriver:

- si on sait décider l'implication logique, la vérification de preuve est décidable
- si on sait construire les invariants, l'inférence faible est décidable.
- si on sait découvrir *l'empreinte*, l'inférence forte est décidable

Les outils existants

- Vérification de preuve:
  Smallfoot, SmallfootRG, HeapHop, Verifast, Dafny, Why,...
- Inférence faible/forte:
  TVLA, TOPICS, L2CA, Xisa... Space Invader :
- Inférence forte:
  Space invader et bi-abduction (autres?)

Quelques questions naturelles

Les classiques

- Décidabilité et complexité de $s, h \models \mathcal{A}$ (MC) et $\models \mathcal{A}$ (SAT)?
- Expressivité des divers fragments

Remarques

- $(s, h) =_L (s', h')$ si graphes isomorphes
- Formule caractéristique $\mathcal{A}_{(s,h)}$ dans $\mathcal{L}$
- (SAT)$\Rightarrow$(MC): $\models \mathcal{A}_{(s,h)} \Rightarrow \mathcal{A}$
- (MC)$\Rightarrow$(SAT) quand on a $\twoheadrightarrow$: $\emptyset, \emptyset \models \neg(\mathcal{A} \twoheadrightarrow \bot)$

Un premier résultat négatif

Un fragment au premier ordre

$\mathcal{A} ::= \quad \neg A \ , \ A \vee A \ , \ \exists x.A \ , \ x \hookrightarrow f1 : y \ , \ x \hookrightarrow f2 : y$

Avec deux sélecteurs: SAT de FO est indécidable (conséquence d'un vieux résultat de Trakhtenbrot).

Encodage de FO relationel

$$
\begin{array}{lcl}
[\![Rxy]\!] & \triangleq & \exists a.(a \hookrightarrow f1 : x, f2 : y) \\
[\![\forall x.\mathcal{A}]\!] & \triangleq & \forall x.[\![\mathcal{A}]\!] \\
[\![\mathcal{A} \ \wedge \ \mathcal{A}']\!] & \triangleq & [\![\mathcal{A}]\!] \ \wedge \ [\![\mathcal{A}']\!]
\end{array}
$$

Séparation et logique du second ordre

La logique SL$-\{*\}$ à un seul sélecteur

$\mathcal{A} ::= \neg A \ , \ A \vee A \ , \ \exists x.A \ , \ A \mathrel{-\!\!*} A \ , \ x \hookrightarrow f : y$

La logique (poly-adique) du second ordre

$\mathcal{A} ::= \neg A \ , \ A \vee A \ , \ \exists x.A \ , \ \exists^2 X.A, X(x_1, \ldots, x_n) \ , \ x \hookrightarrow f : y$

$s, h, \mathcal{E} \models \exists X : n.\mathcal{A}$ si $\exists P \in \mathsf{Val}^n.\ s, h, \mathcal{E}\{X \mapsto P\} \models \mathcal{A}$

Théorème ([Brochenin,CSL'08])

SL$-\{*\}$ a la même expressivité que SO, et les traductions sont effectives.

Séparation et logique du second ordre

Preuve: complexe! cf. exposé de Stéphane.

Corollaires :

- Sur $SL - \{*\}$, SAT est indécidable.
- $SL = SL - \{*\}$, l'étoile est redondante.
- Pourvu que les invariants de boucle s'expriment toujours au second ordre, la logique de séparation est complète pour les programmes séquentiels.

Enfin une bonne nouvelle

La logique SL$-\{-\!\!\!*\}$ à un seul sélecteur

$\mathcal{A} ::= \neg A \ , \ A \vee A \ , \ \exists x.A \ , \ A * A \ , \ x \hookrightarrow f : y$

La logique *monadique* du second ordre

$\mathcal{A} ::= \neg A \ , \ A \vee A \ , \ \exists x.A \ , \ \exists^2 X.A, X(x) \ , \ x \hookrightarrow f : y$

$s, h, \mathcal{E} \models \exists X : n.\mathcal{A}$ si $\exists P \in \mathsf{Val}^n.\ s, h, \mathcal{E}\{X \mapsto P\} \models \mathcal{A}$

Théorème ([Brochenin,CSL'08])

SL$-\{-\!\!\!*\}$ s'encode dans MSO et la traduction est effective.

Conséquence : SAT décidable sur les listes pures [Rabin]

Problème annexe

SL(*) est strictement moins expressif que MSO
[Dawar,FOSSACS'09]

Le parti pri de Smallfoot

Le fragment décidable des « symbolic heaps »

$\mathcal{A} ::= A \vee A \ , \ \exists x.A \ , \ \top, \ A * A, \ , \ x \mapsto f : y \ , \ \mathsf{lseg}(x, y) \ , \ \mathsf{tree}(x)$

Les différences avec ce qui précède:

- plus de négation
- plus de baguette magique
- nombre non borné de sélecteurs
- des prédicats récursifs «built-in»

Expressivité

- $\mathsf{lseg}(x, y)$ s'exprime dans $SL - \{-\!\!*\}$
- en revanche, $\mathsf{tree}(t)$ utilise deux sélecteurs.
- la récursion «générique» se ferait à l'ordre 3.

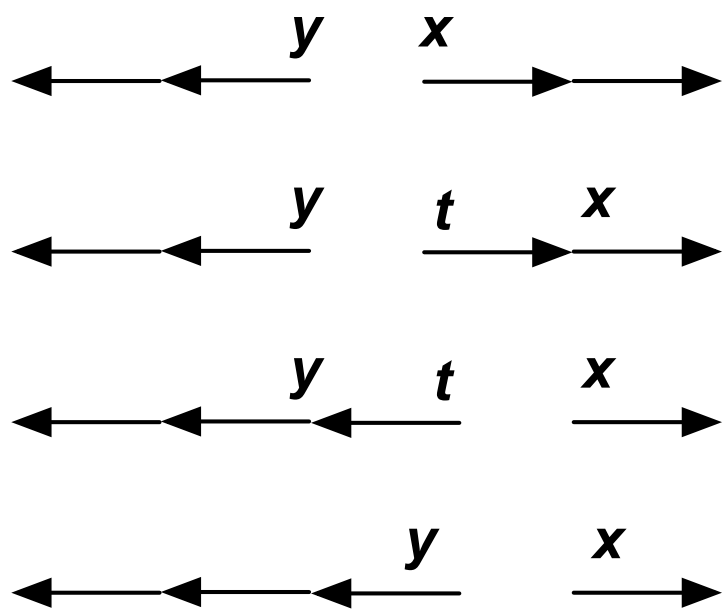
L'analyse de forme (*shape analysis*) .

Principe de l'analyse de forme

- On veut inférer les invariants de boucle, voire l'empreinte.
- On s'intéresse aux propriétés de forme de la mémoire (pas de données).
- Aujourd'hui: on survole le cadre le plus étudié , i.e les programmes avec listes simples.
- Nombreuses techniques pour aller au-delà (automates d'arbres, grammaires de graphes,...)

Reverse and append lists

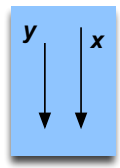
```
function revappend(x,y)
while x<>null do
  t:=x;
  x=[t];
  [t]=y;
  y:=t;
end
```



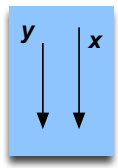
Question

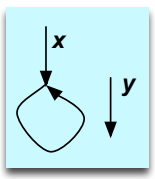
Quelle est la condition nécessaire et suffisante sur la mémoire initiale pour que ce programme termine (sans erreur)? diverge?

L'analyse de formes (shape analysis)

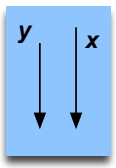


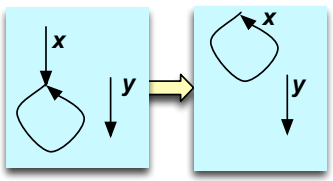
L'analyse de formes (shape analysis)



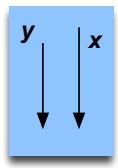


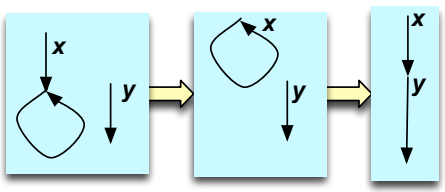
L'analyse de formes (shape analysis)



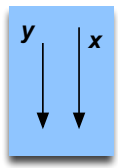


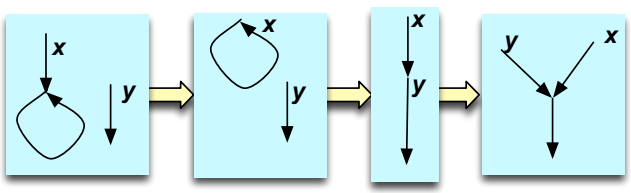
L'analyse de formes (shape analysis)



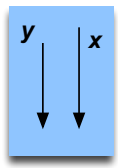


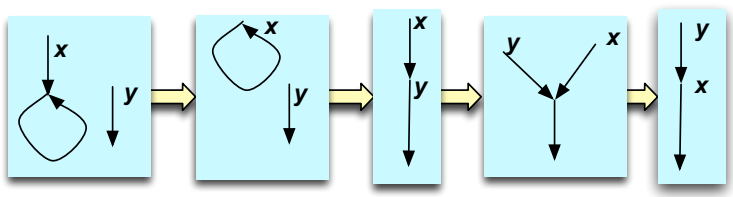
L'analyse de formes (shape analysis)



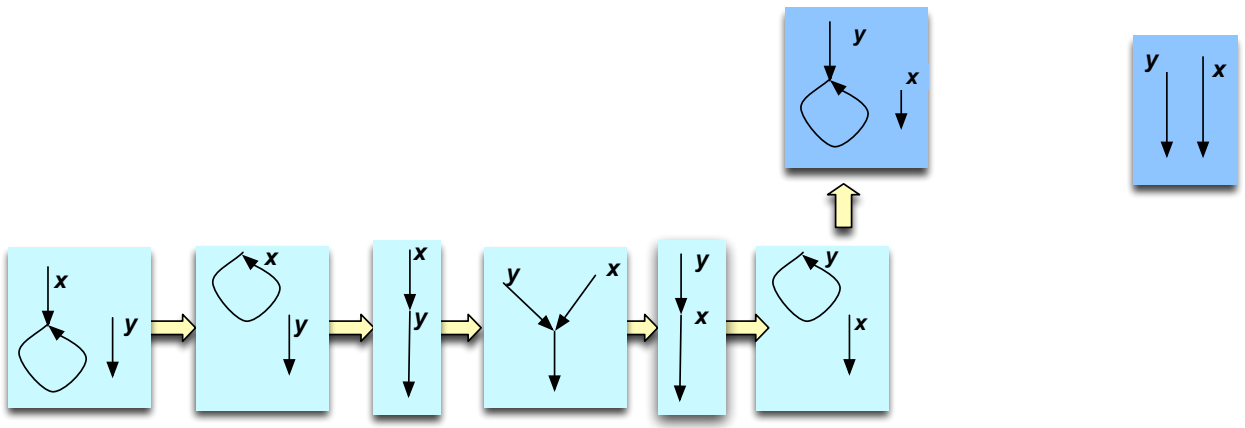


L'analyse de formes (shape analysis)

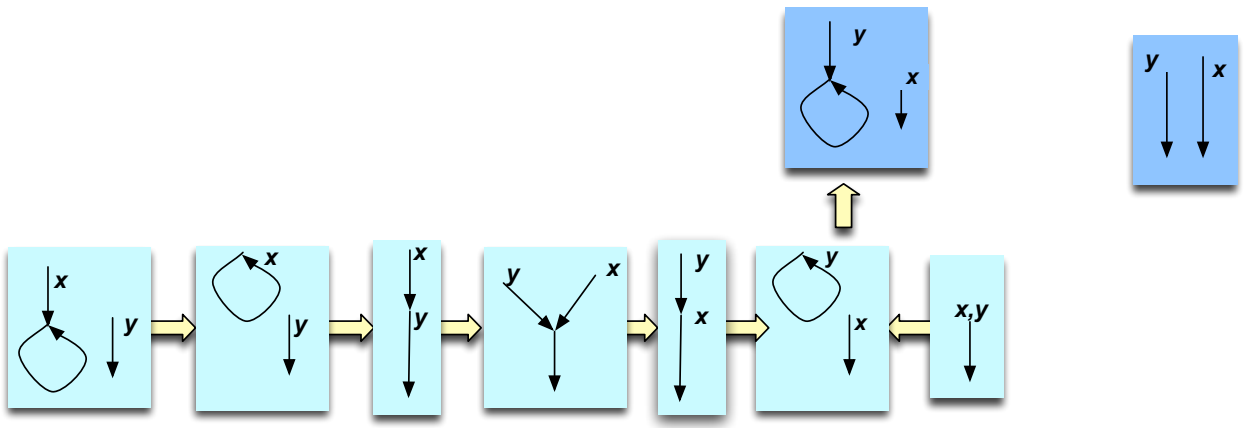




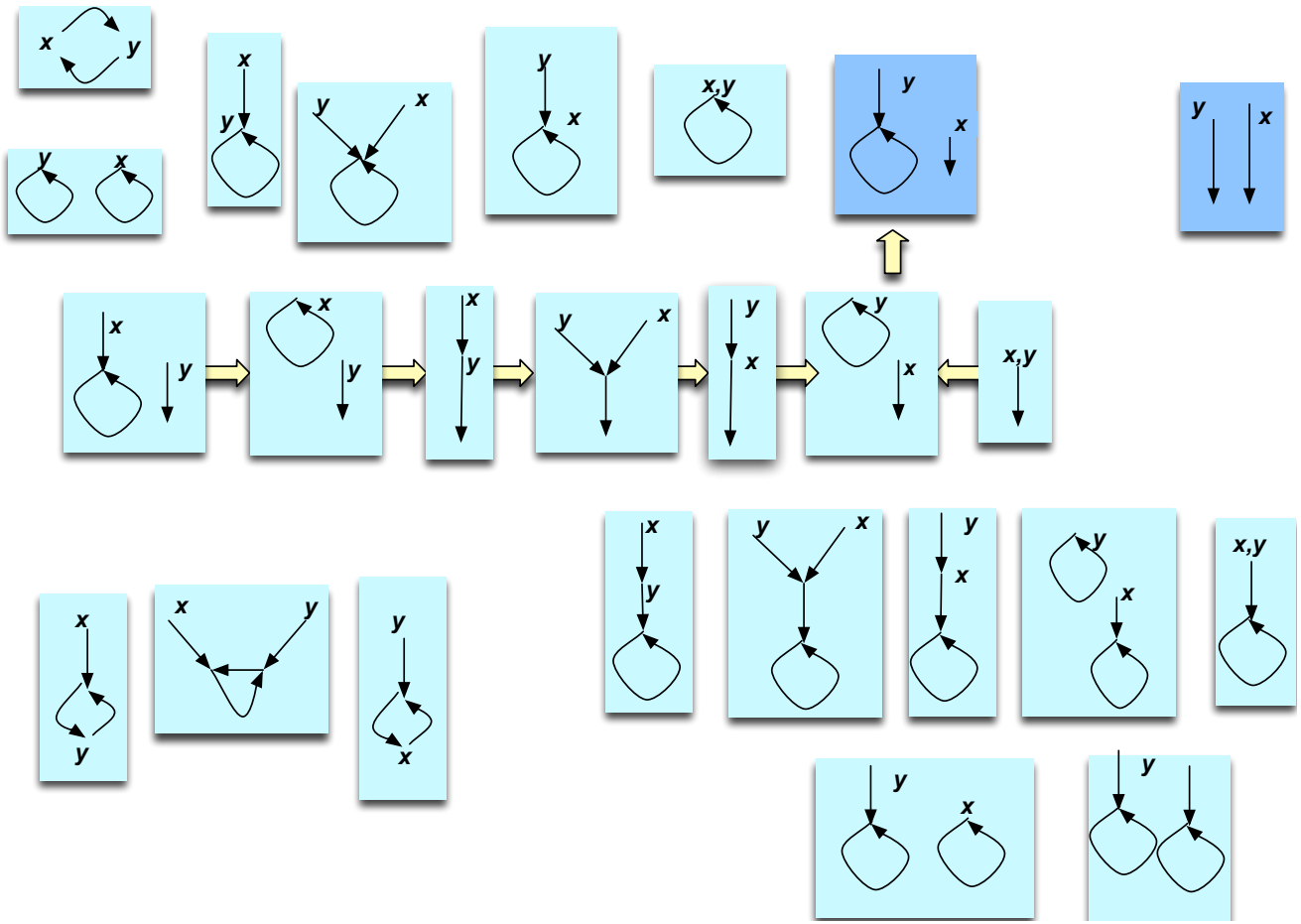
L'analyse de formes (shape analysis)



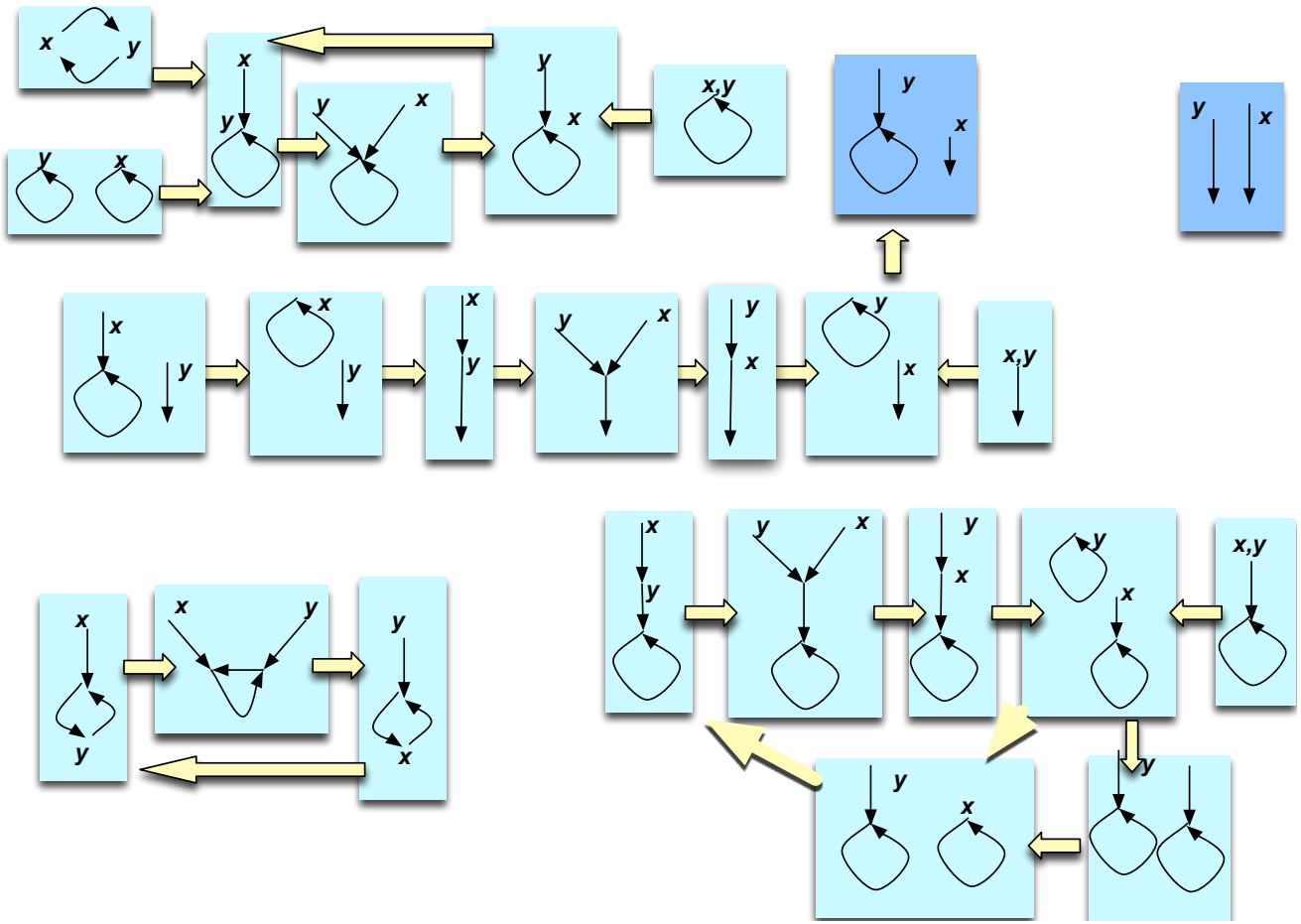
L'analyse de formes (shape analysis)

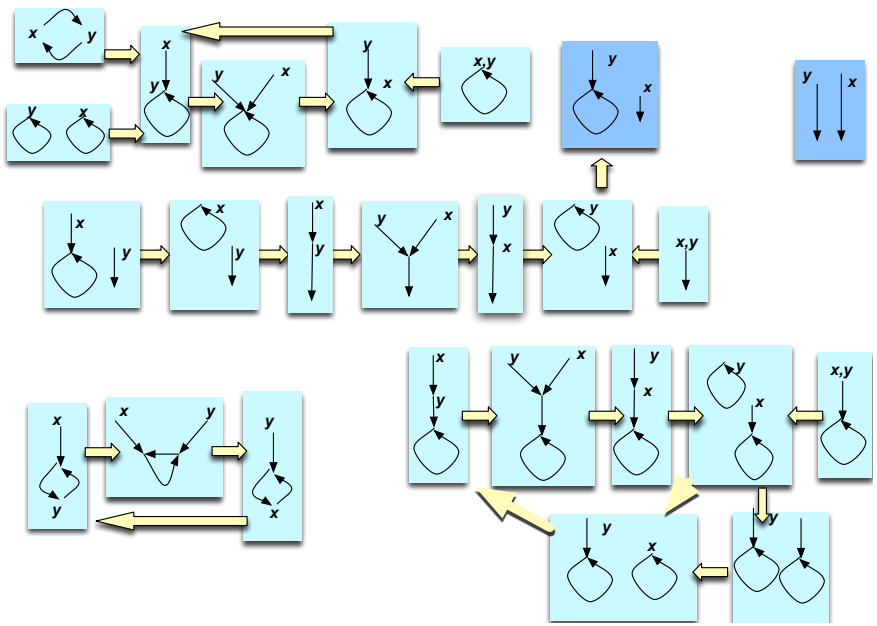


L'analyse de formes (shape analysis)



L'analyse de formes (shape analysis)





Question

Quelle est la condition nécessaire et suffisante sur la mémoire initiale pour que ce programme termine (sans erreur)? diverge?

Réponse

- termine si soit x atteint nul, soit x peut être déréférence infiniment souvent et y atteint null
- diverge si x et y peuvent être déréférencés infiniment souvent

Questions de sémantique .

Monoïde de resources

$\mathcal{M} = (M, \bullet, \varepsilon, \sqsubseteq)$ où :

- $(M, \bullet, \varepsilon)$ est un monoide commutatif
- $(M, \sqsubseteq)$ est un ensemble ordonné
- si $m \subseteq m'$, alors $m \bullet n \subseteq m' \bullet n'$

$m \models A * B$	si il existe $m_1, m_2 . m_1 \models A$, $m_2 \models B$ & $m_1 \bullet m_2 \sqsubseteq m$
$m \models A \mathrel{-\!\!*} B$	pour tout $m' \in M$, si $m' \models A$, alors $m \bullet m' \models B$
$m \models A \rightarrow B$	pour tout $m' \sqsupseteq m$, si $m' \models A$ alors $m' \models B$
$m \models \bot$	jamais

Monoïde de resources

$\mathcal{M} = (M, \bullet, \varepsilon, \sqsubseteq)$ où :

- $(M, \bullet, \varepsilon)$ est un monoide commutatif
- $(M, \sqsubseteq)$ est un ensemble ordonné
- si $m \subseteq m'$, alors $m \bullet n \subseteq m' \bullet n'$

$m \models A * B$	si il existe $m_1, m_2 . m_1 \models A$, $m_2 \models B$ & $m_1 \bullet m_2 \sqsubseteq m$
$m \models A \mathrel{-\!\!*} B$	pour tout $m' \in M$, si $m' \models A$, alors $m \bullet m' \models B$
$m \models A \rightarrow B$	pour tout $m' \sqsupseteq m$, si $m' \models A$ alors $m' \models B$
$m \models \bot$	jamais

Problème: $A \mathrel{-\!\!*} \bot$ n'est pas traité pareil dans la preuve et la sémantique

La solution de Méry

Monoïde de resources

$\mathcal{M} = (M, \bullet, \varepsilon, \sqsubseteq)$ où :

- $(M, \bullet, \varepsilon)$ est un monoide commutatif
- $(M, \sqsubseteq)$ est un ensemble ordonné
- si $m \subseteq m'$, alors $m \bullet n \subseteq m' \bullet n'$
- $\bullet$ est **partiel** : $m \bullet m'$ n'est pas forcément défini, mais quelques contraintes:
  - $m \bullet \varepsilon = m$;
  - si $m \bullet m'$ est défini, alors $m' \bullet m$ est défini;
  - si $m \bullet m'$ et $(m \bullet m') \bullet m''$ sont définis, alors $m' \bullet m''$ et $m \bullet (m' \bullet m'')$ sont définis.

La solution de Méry

Monoïde de resources

$\mathcal{M} = (M, \bullet, \varepsilon, \sqsubseteq)$ où :

- $(M, \bullet, \varepsilon)$ est un monoide commutatif
- $(M, \sqsubseteq)$ est un ensemble ordonné
- si $m \subseteq m'$, alors $m \bullet n \subseteq m' \bullet n'$
- $\bullet$ est **partiel** : $m \bullet m'$ n'est pas forcément défini, mais quelques contraintes:
  - $m \bullet \varepsilon = m$;
  - si $m \bullet m'$ est défini, alors $m' \bullet m$ est défini;
  - si $m \bullet m'$ et $(m \bullet m') \bullet m''$ sont définis, alors $m' \bullet m''$ et $m \bullet (m' \bullet m'')$ sont définis.

Autre solution : les Grothendieck resource model, plus compliqué, pas très intuitif non plus.

La solution de Méry

Monoïde de resources

$\mathcal{M} = (M, \bullet, \varepsilon, \sqsubseteq)$ où :

- $(M, \bullet, \varepsilon)$ est un monoide commutatif
- $(M, \sqsubseteq)$ est un ensemble ordonné
- si $m \subseteq m'$, alors $m \bullet n \subseteq m' \bullet n'$
- $\bullet$ est **partiel** : $m \bullet m'$ n'est pas forcément défini, mais quelques contraintes:
  - $m \bullet \varepsilon = m$;
  - si $m \bullet m'$ est défini, alors $m' \bullet m$ est défini;
  - si $m \bullet m'$ et $(m \bullet m') \bullet m''$ sont définis, alors $m' \bullet m''$ et $m \bullet (m' \bullet m'')$ sont définis.

Autre solution : les Grothendieck resource model, plus compliqué, pas très intuitif non plus.
Plus général [Méry]: $m = m_1 \bullet m_2$ est une relation ternaire qui...

On enlève la partie intuitioniste:

Algèbre de séparation

$\mathcal{M} = (M, \bullet, \varepsilon)$ est un monoide : partiel, commutatif, et intègre: si $m_1 \bullet m = m_2 \bullet m$, alors $m_1 = m_2$.

$m \models A * B$	si il existe m_1, m_2. $m_1 \models A$, $m_2 \models B$ &$m_1 \bullet m_2 = m$
$m \models A \mathrel{-\!\!*} B$	pour tout $m' \in M$, si $m' \models A$, alors $m \bullet m' \models B$
$m \models A \to B$	si$m \models A$alors $m \models B$
$m \models \bot$	jamais

Quelques définitions

- le préordre de division: $m' \leq m$ si il existe m'' tel que $m = m' \bullet m''$
- la cohérence: $m \bot m'$ (notation standard!) si $m \bullet m'$ est défini.
- la septraction: m'' est unique, noté $m - m'$.

Exemples d'algèbres de séparation

Les configurations des réseaux de Petri:
$(\mathbb{Z}^P, +, \vec{0})$, où P est l'ensemble des places.

Le modèle mémoire «rudimentaire»
$Loc \rightharpoonup Val, \cup, \emptyset)$, où Loc=Val+$\mathbb{N}$.

Le modèle mémoire avec permissions:
$(Loc \rightarrow (Perm \times Val), \bullet, \emptyset)$, où $(Perm, \bullet, \varepsilon)$, est une algèbre de séparation.

$h_1 \bullet h_2 : l \mapsto (p_1 \bullet p_2, v)$ si $h_i(l) = (p_i, v)$ et $p_1 \perp p_2$.

Permissions, qu'est-ce que c'est?

Algèbre de permission:
Un tuple $(M, \bullet, \varepsilon, \top)$ tel que $(M, \bullet, \varepsilon)$ est une algèbre de séparation et $\varepsilon, \top$ sont respectivement min et max pour $\leq$.

Intuition:

- ε représente la permission nulle : on ne peut ni lire ni écrire
- $\top$ représente la permission totale: on peut lire et écrire
- tout le reste représente un permission en lecture seule

Trois exemples, on reverra plus loin:

- Les permissions grossières : $(\{0, 1\}, +)$.
- Les permissions fractionnaires de Boyle: $([0, 1], +)$.
- Les permissions entières de Parkinson: $(\mathcal{P}_c(\mathbb{N}), \uplus)$, i.e. les parties finies et cofinies de $\mathbb{N}$ avec union disjointe.

Les fonctions locales

Le domaine des résultats: si $(\Sigma, \bullet, \varepsilon)$ est une S.A., alors $(\mathcal{P}(\Sigma)^\top, \sqsubseteq)$ est l'ensemble $\mathcal{P}(\Sigma)$ ordonné par $\subseteq$, plus un élément maximal $\top$ au-dessus de Σ.
$(\mathcal{P}(\Sigma)^\top, *, emp)$ définit un monoide commutatif , en posant $\top * A = \top$.
Intuition de $\top$: état d'erreur.

Fonction locale: $f : \Sigma \to \mathcal{P}(\Sigma)^\top$ est locale si

$$f(m \bullet m') \sqsubseteq f(m) * \{m'\}.$$

Intuition: si le programme a assez de ressources pour ne pas fauter, alors avec toute ressource supplémentaire il ne faute pas non plus et a un comportement simulable par «laisser la ressource supplémentaire intacte».

Correction de la frame rule

Triplets de Hoare pour $A, B \in \mathcal{P}(\Sigma)$, on note $\{A\}\ \mathsf{f}\ \{B\}$ si $f(A) \sqsubseteq B$, i.e. tous les états de A sont sûrs et mènent à B.

Si f est locale:

$$\frac{\{\mathcal{A}\}\ f\ \{\mathcal{B}\}}{\{\mathcal{A} * \mathcal{F}\}\ f\ \{\mathcal{B} * \mathcal{F}\}}$$

De la sem. axiomatique à la dénotationnelle

Spécification: un ensemble de couples pre/post, $\mathcal{S} = \{(A_i, B_i)\}_{i \in I}$ dans $\mathcal{P}(\Sigma)$.

Best Local Action:

$$bla[\mathcal{S}] = m \mapsto \bigsqcap_{i \in I, m' \in A_i, m' \leq m} B_i * \{m - m'\}$$

Remarque: c'est ici qu'on a besoin d'avoir $(\Sigma, \bullet)$ *intègre*.

Propriété de $bla[\mathcal{S}]$:

1. c'est une fonction locale
2. elle satisfait $\mathcal{S}$
3. c'est la plus grande fonction vérifiant 1 et 2 pour l'ordre «point à point» sur les fonctions.

Exemple: la sémantique de l'allocation mémoire

Spécification de new par small axioms:
$\{x \mapsto -\}$ x:= new() $\{\exists l.x \mapsto l * l \mapsto -\}$.

Sémantique dénotationnelle de new:
si $s(x)$ est défini,

$$[\![x := new()]\!](s,h) = \bigcup \{(s[x \mapsto l], h \bullet l \mapsto v) \ : \ l \notin domh\}$$

sinon $[\![x := new()]\!](s,h) = \top$ (la variable x n'est pas déclarée).

A noter:

- la condition de fraîcheur découle de la localité
- pour cette fonction, $f(m_1 \bullet m_2) \sqsubset f(m_1) * \{m_2\}$ même si $f(m_1) \neq \top$

Propriétés des fonctions locales

Composition séquentielle $f, g : \Sigma \to \mathcal{P}(\Sigma)^\top$ se composent en

$$f ; g(m) = \bigsqcup_{m' \in f(m)} g(m')$$

Remarque: on peut aussi définir union $f + g$, et itération f^*.

Composition parallèle (version simplifiée):
$f || g : m \mapsto \bigsqcap_{m_1 \bullet m_2 = m} f(m_1) * g(m_2)$

Si Σ est *cross-split* (voir après), alors $f||g$ est locale.

Correction de la règle de disjoint concurrency

Composition parallèle (VO):
sémantique d'interleaving: f et g sont des séquences d'actions atomiques, tout interleaving définit la même fonction, qui faute si il y a une race.

Correction de la Disjoint Concurrency's Rule:

$$\frac{\{\mathcal{A}_1\}\ P_1\ \{\mathcal{B}_1\} \quad \{\mathcal{A}_2\}\ P_2\ \{\mathcal{B}_2\}}{\{\mathcal{A}_1 * \mathcal{A}_2\}\ P_1 || P_2\ \{\mathcal{B}_1 * \mathcal{B}_2\}}$$

La notion d'empreinte (Raza& Gardner, FOSSACS'0

Empreinte: utile pour l'inférence de preuve forte.

Deux intuitions de la notion d'empreinte:

- l'ensemble des plus petits états qui garantissent une non-faute
- l'ensemble des états qui vérifient au moins une des préconditions

Divergence des intuitions: AD $\stackrel{\triangle}{=}$ x:=new();dispose x .

- a pour plus petit état sûr $\mathbf{x} \mapsto -$
- a pour spécification complète:

$$\begin{array}{ccc} \{x \mapsto -\} & AD & \{x \mapsto -\} \\ \{x \mapsto - * l \mapsto -\} & AD & \{x \mapsto l' * l \mapsto - * l \neq'\} \end{array}$$

Quelle est l'empreinte de AD??

La bonne notion d'empreinte

Définition axiomatique: Si $\mathcal{S} = \{(A_i, B_i)\}_{i \in I}$ est une spécification,

$$\text{footprint}(\mathcal{S}) = \bigsqcup_{i \in I} A_i$$

Définition sémantique[Raza]: Si f est une fonction locale, $m \in \text{footprint}(f)$ si

$$f(m) \sqsubset \bigsqcap_{m' < m} f(m') * \{m - m'\}$$

Intuition: les «points de discontinuité».

Théorème:

$$\text{footprint}(bla[\mathcal{S}]) = \text{footprint}(\mathcal{S})$$

La bonne notion d'empreinte

Définition axiomatique: Si $\mathcal{S} = \{(A_i, B_i)\}_{i \in I}$ est une spécification,

$$\text{footprint}(\mathcal{S}) = \bigsqcup_{i \in I} A_i$$

Définition sémantique[Raza]: Si f est une fonction locale, $m \in \text{footprint}(f)$ si

$$f(m) \sqsubset \bigsqcap_{m' < m} f(m') * \{m - m'\}$$

Intuition: les «points de discontinuité».

Théorème:

$$\text{footprint}(bla[\mathcal{S}]) = \text{footprint}(\mathcal{S})$$

Pour aller plus loin: notion de base, toute base est empreinte, conditions pour avoir égalité, cf Raza&Garnder,FOSSACS'08.

Les algèbres de séparation revisitées

Dockins&Hobor&Appel, APLAS'09

Algèbres de séparation multi-unitaires

$\mathcal{M} = (M, \bullet)$ où $\bullet$ est une loi de composition interne:

- partielle
- commutative
- intègre

Unités: $m \models emp$ si $m^2 = m$

Tout le reste marche comme avant!

Les algèbres de séparation revisitées

Dockins&Hobor&Appel, APLAS'09

Algèbres de séparation multi-unitaires

$\mathcal{M} = (M, \bullet)$ où $\bullet$ est une loi de composition interne:

- partielle
- commutative
- intègre

Unités: $m \models emp$ si $m^2 = m$

Tout le reste marche comme avant!

Utilité principale: on équipe la catégorie des SA d'un co-produit, et n'importe quel ensemble admet une structure naturelle de SA discrète.

Les algèbres de séparation revisitées

Dockins&Hobor&Appel, APLAS'09

Algèbres de séparation multi-unitaires

$\mathcal{M} = (M, \bullet)$ où $\bullet$ est une loi de composition interne:

- partielle
- commutative
- intègre

Unités: $m \models emp$ si $m^2 = m$

Tout le reste marche comme avant!

Utilité principale: on équipe la catégorie des SA d'un co-produit, et n'importe quel ensemble admet une structure naturelle de SA discrète.
Il parait que ça se fait aussi dans les relevant logics.

Propriétés supplémentaires

Positivité : on ne peut pas séparer une unité en autre chose qu'elle même.
Formulations équivalentes:

- $m_1 \bullet m_2 = m \Rightarrow m \bullet m = m \Rightarrow m_1 \bullet m_1 = m_1$
- $m_1 \bullet m_2 = m \Rightarrow m \bullet m = m \Rightarrow m_1 \bullet m_1 = m_2 = m$
- le préordre de division est un ordre (i.e. anti-symétrique).

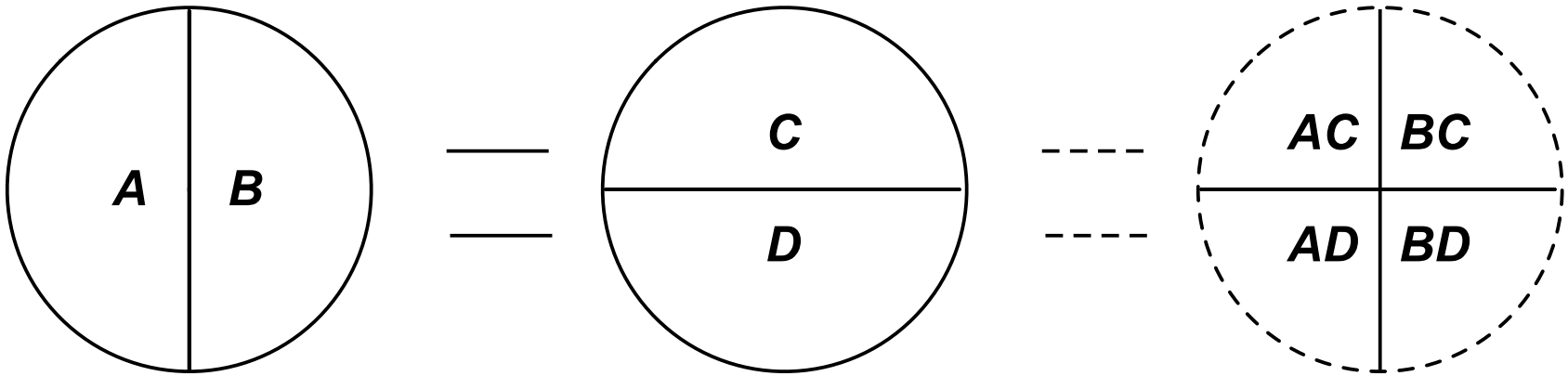
Super-positivité («disjointness»): toute resource non-unité est incompatible avec elle-même.
Formulations équivalentes:

- $a \bullet a = b$ implique $a = b$
- si $a \perp b$, alors $\sup\{a, b\}$ existe et vaut $a \bullet b$.

Remarque: super-positif implique positif.

Propriétés supplémentaires

Cross-splitting:



Infinite splitting: $\neg\mathsf{emp} \vdash \neg\mathsf{emp} * \neg\mathsf{emp}$.

Morale:

- ces propriétés peuvent ou non être utiles (ex: ma def de $f||g$);
- elles ont le bon goût d'être préservées par produit et co-produit.

La concurrence et le partage .

Les moniteurs de Hoare

Syntaxe: P::= ... | with r when b do P

- r est un identifiant de ressource
- b est une condition booléene
- P est un programme

Annotation requise: pour chaque ressource r, un invariant I_r.

Règle associée:

$$\frac{\{b \wedge (\mathcal{A} * I_r)\}\ \mathsf{P}\ \{\mathcal{B} * I_r\}}{\{\mathcal{A}\}\ \mathsf{with}\ \ \mathsf{r}\ \ \mathsf{when}\ \mathsf{b}\ \mathsf{do}\ \mathsf{P}\ \{\mathcal{B}\}}$$

Exemples

- pointer non transferring buffer
- pointer transferring buffer
- pointer transferring buffer no leak

Les échanges de messages

Version simplifiée de Heaphop (un seul canal)
$P ::= \ldots \mid send(m, x) \mid x = receive(m)$

Annotation requise: Un invariant de message I_m pour chaque identifiant de message m.

Règles associées:

$\{I_m\}$ send(m,E) $\{emp\}$ $\qquad$ $\{emp\}$ x := receive(m) $\{I_m\}$

Les échanges de messages à la Heaphop

Chaque (bi)canal est accessible par deux extremités
$P ::= \ldots \mid (e, f) = open(C) \mid close(e, f) \mid send(m, e, x) \mid x = receive(m, e)$

Annotation requise: contrat C (cf. exposé Jules).

Exemple:

```
local e,f in
  (e,f)=open(C);
  send(m,e,a);
  b:=receive(m,f);
  close(e,f);
```

$\approx$

```
b:=a;
```

Les règles utilisées dans Heaphop

$$\frac{i = \mathrm{init}(C)}{\{\mathsf{emp}\}\ (\mathsf{e},\mathsf{f}){:=}\mathsf{open}(\mathsf{C})\ \{e \overset{peer}{\mapsto} (C[i], f) * f \overset{peer}{\mapsto} (\bar{C}[i], e)\}}$$

$$\frac{f \in \mathrm{final}(C)}{\{E \overset{peer}{\mapsto} (C[f], E') * E' \overset{peer}{\mapsto} (\bar{C}[f], E)\}\ \mathsf{close}\,(\mathsf{E},\mathsf{E'})\ \{\mathsf{emp}\}}$$

Les règles utilisées dans Heaphop

$$\frac{a \xrightarrow{?m[l]} b \in C}{\{E \overset{peer}{\mapsto} (C[a], X)\}\ \mathsf{x} := \mathsf{receive}\,(\mathsf{E.m})\ \{E \overset{peer}{\mapsto} (C[b], X) * l(x, X)\}}$$

Les règles utilisées dans Heaphop

$$\frac{a \xrightarrow{!m[l]} b \in C}{\{E \overset{peer}{\mapsto} (C[a], -) * I(E', E)\}\ \mathsf{send}(\mathrm{E.m,E'})\ \{E \overset{peer}{\mapsto} (C[b], -)\}}$$

$$\frac{a \xrightarrow{!m[l]} b \in C}{\{E \overset{peer}{\mapsto} (C[a], -) * (E \overset{peer}{\mapsto} (C[b], -) * I(E', E))\}\ \mathsf{send}(\mathrm{E.m,E'})\ \{O || - \mathrm{emp}\}}$$

Exemples

- list transfer
- load balancing parallel tree disposal

Relacher l'exclusion mutuelle

Concurrence non-bloquante: Rely/Guarantee et SL (cf. Vafeiadis, SmallfootRG)

Pour aujourd'hui, moins ambitieux: Lecteurs multiples, rédacteur unique.

Exemple:

```
[list(x)]
   search(x,v1) || search(x,v2)
[list(x)]
```

Les permissions fractionnaires

Un nouveau prédicat: $x \overset{p}{\mapsto} y$, où $p \in [0, 1]$.

Axiomatique:

$$x \overset{p}{\mapsto} y * x \overset{q}{\mapsto} z \vdash x \overset{p+q}{\mapsto} y \wedge y = z$$

Liste partagée: $\mathit{list}(x, p) = (\mathsf{emp} \wedge x = 0) \vee \exists x'.x \overset{p}{\mapsto} x' * \mathit{list}(x', p)$

Axiomatique

$$\{x \overset{1}{\mapsto} -\}\ x->f := v\ \{x \overset{1}{\mapsto} v\}$$
$$\{x \overset{p}{\mapsto} -\}\ y := x->f\ \{x \overset{p}{\mapsto} y\} \quad (\text{avec } p > 0)$$

Le problème de l'arbre partagé

La définition intuitive:

$$tree(x, p) = \text{if } (x = 0) \text{ then emp}$$
$$\text{else} \exists x', x''.x \overset{p}{\mapsto} x', x'' * tree(x', p) * tree(x'', p)$$

Problème[Parkinson] : on autorise aussi les DAG!

Les permissions entières

Un nouveau prédicat: $x \overset{p}{\mapsto} y$, où $p \in \mathbb{Z}$.

Intuition:

- $p = 0$: permission totale.
- $p < 0$ (en général, $p = -1$): on a *reçu* p permissions en lecture
- $p > 0$: on a *donné* p permissions en lecture

Axiomatique:

$$x \overset{p+q}{\mapsto} v * x \overset{-q}{\mapsto} v \dashv\vdash x \overset{p}{\mapsto} v \qquad (p \geq 0, q > 0)$$

$$x \overset{-(p+q)}{\mapsto} v \dashv\vdash x \overset{-p}{\mapsto} v * x \overset{-q}{\mapsto} v \qquad (p, q > 0)$$

$$x \overset{0}{\mapsto} v * x \overset{p}{\mapsto} \dashv\vdash \bot$$

Remarques: on a un seul distributeur/collecteur de permissions (semble généralisable...)

Le modèle de Parkinson (suite)

Axiomatique:

$$x \overset{p+q}{\mapsto} v * x \overset{-q}{\mapsto} v \quad \dashv\vdash \quad x \overset{p}{\mapsto} v \qquad (p \geq 0, q > 0)$$

$$x \overset{-(p+q)}{\mapsto} v \quad \dashv\vdash \quad x \overset{-p}{\mapsto} v * x \overset{-q}{\mapsto} v \qquad (p, q > 0)$$

$$x \overset{0}{\mapsto} v * x \overset{p}{\mapsto} \quad \dashv\vdash \quad \bot$$

Algèbre de permission: $(\mathcal{P}_c(\mathbb{N}), \uplus)$, i.e. les parties finies et cofinies de $\mathbb{N}$ avec union disjointe.

Sémantique: pour $N \in \mathcal{P}_c(\mathbb{N})$,

$N \models -p$ si $\#N = p$
$N \models p$ si $\complement N \models -p$
$N \models 0$ si $N = \mathbb{N}$
$N \models \mathrm{emp}$ si $N = \emptyset$.

Exemple: les lecteurs et redacteurs

LECTEURS

```
with read do if count = 0
then lock(write); count++;
done

...:= a->f

with read when count>0 do
count--; if count = 0 then
unlock(write); done
```

REDACTEURS

```
lock(write);

a->f :=...

unlock(write)
```

Encodage du verrou

```
lock(write): with write when s > 0 do s := s-1
unlock(write): with write when true do s := s+1
```

Invariants de ressource:

ressource write $\{\text{if } s = 0 \text{ then emp else } a \overset{0}{\mapsto} -\}$

ressource read $\{\text{if } \mathit{count} = 0 \text{ then emp else } * a \overset{\mathit{count}}{\mapsto} -\}$

Pre/Post des opérations de verrous

$\{\text{emp}\}$	$\mathit{lock}(\mathit{write})$	$\{a \overset{0}{\mapsto} -\}$
$\{a \overset{0}{\mapsto} -\}$	$\mathit{unlock}(\mathit{write})$	$\{\text{emp}\}$

Preuve (suite)

LECTEURS

$\{emp\}$

```
with read do if count = 0
then lock(write); count++;
done
```

$\{a \overset{-1}{\mapsto} -\}$

```
...:= a->f
```

$\{a \overset{-1}{\mapsto} -\}$

```
with read when count>0 do
count--; if count = 0 then
unlock(write); done
```

$\{emp\}$

REDACTEURS

$\{emp\}$

```
lock(write);
```

$\{a \overset{0}{\mapsto} -\}$

```
a->f :=...
```

$\{a \overset{0}{\mapsto} -\}$

```
unlock(write)
```

$\{emp\}$