Pointers static analysis and BI-logic

Mémoire de D.E.A.
École Polytechnique, France
April-August 2002

Advisor : Radhia Cousot

Élodie-Jane Sims.

Under construction

Latest version : `http://www.eleves.ens.fr:8080/home/sims/download/DEA/rapport-stage.ps.gz`

This document has colors.

Contents

Chapter 1

Introduction

Mon stage se place dans le cadre de l'analyse statique de logiciels comportant des objets alloués dynamiquement sur un tas et repérés par des pointeurs. Samin Ishtiaq, Peter O'Hearn et John Reynolds ont développé récemment la *logique BI* [4] qui est une logique de Hoare avec un langage d'assertions/de prédicats permettant de démontrer qu'un programme manipulant des pointeurs sur un tas est correcte. Nous souhaiterions utiliser la logique BI comme interface pour exprimer les résultats d'analyses modulaires de pointeurs.

Pendant mon stage, j'ai étudié la logique BI que je présente au *chapitre 2*. J'ai prouvé la correction des plus faible pré-conditions (wlp)*(chapitre 3)*. J'ai exprimé les plus fortes post-conditions (sp), et conditions d'absence d'erreurs ($wlp(\texttt{true}, C)$) et prouvé la correction de l'analyse avant *(chapitre 4)*. J'ai ensuite démontré la correction de l'analyse de pointeurs en avant [2] en exprimant une traduction des partitions (invariants de l'analyse) dans le langage BI *(chapitre 5)*.

The aim of this starting work is to do static analysis on programs dealing with pointers. Developed by Ishtiaq, O'Hearn, Pym, Reynolds and Yang, BI-logic [4] is a Hoare logic for reasoning about properties of heap storage, in terms of Hoare triples, $\{P\}C\{P'\}$. We would like to use BI-logic as an interface language for modularity with other pointers analyzes.

We present the logic BI in *chapter 2*. We proved the correction of the weakest preconditions (wlp) *(chapter 3)*. We have expressed the strongest postconditions (sp) and the conditions of safe executions ($wlp(\texttt{true}, C)$) and proved the correction of the forward analysis *(chapter 4)*. Then we proved the correction of the pointers forward analysis [2] by expressing the translation of partitions (analysis's invariants) into BI's language *(chapter 5)*.

Chapter 2

BI-logic

2.1 Introduction

O'Hearn and Pym have developed the logic BI of bunched[1] implications[5]. This is a Hoare logic with an assertions/predicates language for reasoning about properties of heap storages, in terms of Hoare triples, $\{P\}C\{P'\}$. It allows to prove that a program dealing with pointers on a heap is correct[6, 4]. This chapter gives a presentation of BI that comes from [4]. BI-logic is a classical logic over states memory with some specific spatial connectives $*$ and $-\!\!*$. $P * Q$ asserts that P and Q holds for separate parts of a data structure. It provides a way to compose assertions that refer to different areas of memory.

In this chapter, we first present the language we are analyzing, then the BI language itself and then the Hoare's triplets of the analysis.

2.2 Commands

We deal with the usual commands of an imperative language with some restrictions :

- we do not directly handle double-dereferencing, such as $x.i.j$
- dereferencing should be avoided either on the left or right of :=
- a pointer points only to a pair of cells.

Those restrictions do not limit the expressive power of the language.

2.2.1 Syntax

$$
\begin{array}{rcc}
C & ::= & x := E \\
& | & x := E.i \\
& | & E.i := E' \\
& | & x := \texttt{cons}(E_1, E_2) \\
& | & \texttt{dispose}(E) \\
& | & C_1; C_2 \\
& | & if\ E\ then\ C_1\ else\ C_2 \\
& | & if\ E\ then\ C_1 \\
& | & while\ E\ do\ C_1
\end{array}
\qquad
i ::= 1|2
\qquad
\begin{array}{rcll}
E & ::= & x & Variable \\
& | & 42 & Integer \\
& | & \texttt{nil} & Atom \\
& | & a & Atom \\
& | & True & Atom \\
& | & False & Atom \\
& | & E1\ op_a\ E_2 & Arithmetic\ operation \\
& | & E1\ op_b\ E_2 & Boolean\ operation
\end{array}
$$

[1] The name comes from the sequent calculus of BI, instead of having $\Gamma \vdash A$ where Γ is only a list of propositions like in $A\ ;\ B \vdash A \wedge B$, we also have "bunches" like in $A\ ,\ B \vdash A * B$,

2.2.2 Configurations

Domain : A command is executed over a stack $s \in S$ and a heap $h \in H$

$$\begin{array}{rcl} Val & = & Int \cup Atoms \cup Loc \\ S & = & Var \rightharpoonup_{fin} Val \\ H & = & Loc \rightharpoonup_{fin} Val \times Val \end{array}$$

Val = set of values
Loc = $\{l, ...\}$ is an infinite set of locations
Var = $x, y, ...$ is a set of variables
$Atoms$ = $\texttt{nil}, a, ...$ is a set of atoms
$\rightharpoonup_{fin}$ is for finite partial functions.

Configuration : A configuration is either C, s, h a command C to be executed on a memory s, h, either a *terminal configuration* s, h.
Definitions :

- C, s, h **stuck** : $\nexists K\ C, s, h \rightsquigarrow K$
- C, s, h **safe** : If $C, s, h \rightsquigarrow^* K$ Then K is not stuck.

Notice that we call *safe configuration* (among others) all the configurations with infinite computation.

2.2.3 Operational semantics : $C, s, h \rightsquigarrow s', h' \quad C, s, h \rightsquigarrow C', s', h'$

The operational semantics requires the definition of the semantics of an expression $[\![E]\!]s$.

$$\begin{array}{rcl} [\![x]\!]s & = & s(x) \\ [\![42]\!]s & = & 42 \\ [\![\texttt{true}]\!]s & = & true \\ [\![E_1 + E_2]\!]s & = & [\![E_1]\!]s + [\![E_2]\!]s \\ & ... & \end{array}$$

Notations In the following rules we use :

- r to range over elements of $Val \times Val$
- $\pi_i r$ for the first or second projection
- $(r|i \mapsto v)$ to indicate the pair like r except that the i'th component is replaced with v
- $[f|x \mapsto v]$ to indicate the function like f except that it maps x to v
- $h - l$ is the heap like h except that it is undefined on l

Stack-altering commands

$$\frac{[\![E]\!]s = v}{x := E, s, h \rightsquigarrow [s|x \mapsto v], h}$$

assign to x in the stack the value of E

$$\frac{[\![E]\!]s = l \in Loc \quad h(l) = r}{x := E.i, s, h \rightsquigarrow [s|x \mapsto \pi_i r], h}$$

assign to x in the stack the value of the i'th component of $[\![E]\!]s$ in the heap

Heap-altering commands

$$\frac{[\![E]\!]s = l \in Loc \quad h(l) = r \;\; [\![E']\!]s = v'}{E.i := E', s, h \rightsquigarrow s, [h|l \mapsto (r|i \mapsto v')]}$$

assign in the heap to the i'th component of $[\![E]\!]s$ *the value of* E'

$$\frac{l \in Loc \quad l \in dom(h) \;\; [\![E]\!]s = l}{\texttt{dispose}(E), s, h \rightsquigarrow s, (h - l)}$$

free in the heap the location value of E
notice that if E *is the variable it creates a dangling pointer since the stack in not modified*

Stack- and heap-altering commands

$$\frac{l \in Loc \;\; l \notin dom(h) \;\; [\![E_1]\!]s = v_1, [\![E_2]\!]s = v_2}{x := \texttt{cons}(E_1, E_2), s, h \rightsquigarrow [s|x \mapsto l], [h|l \mapsto \langle v_1, v_2 \rangle]}$$

assign to x *in the stack a fresh location and assign in the heap to this fresh location the two values of* E_1 *and* E_2

Composed commands

$$\frac{C_1, s, h \rightsquigarrow C', s', h'}{C_1; C_2, s, h \rightsquigarrow C'; C_2, s', h'}$$

$$\frac{C_1, s, h \rightsquigarrow s', h'}{C_1; C_2, s, h \rightsquigarrow C_2, s', h'}$$

$$\frac{[\![E]\!]s = True}{if \;\; E \;\; then \;\; C_1 \;\; else \;\; C_2, s, h \rightsquigarrow C_1, s, h}$$

$$\frac{[\![E]\!]s = False}{if \;\; E \;\; then \;\; C_1 \;\; else \;\; C_2, s, h \rightsquigarrow C_2, s, h}$$

$$\frac{[\![E]\!]s = True}{if \;\; E \;\; then \;\; C_1, s, h \rightsquigarrow C_1, s, h}$$

$$\frac{[\![E]\!]s = False}{if \;\; E \;\; then \;\; C_1, s, h \rightsquigarrow s, h}$$

$$\frac{[\![E]\!]s = True}{while \;\; E \;\; do \;\; C, s, h \rightsquigarrow C; while \;\; E \;\; do \;\; C, s, h}$$

$$\frac{[\![E]\!]s = False}{while \;\; E \;\; do \;\; C, s, h \rightsquigarrow s, h}$$

Notice that $\rightsquigarrow$ is a not a total function.

2.3 Formulas

P, Q, R	::=	α	Atomic formula						
	|	$\texttt{false}$							
	|	$P \Rightarrow Q$	Classical implication	α	::=	$E = E'$	E	::=	x $\quad Var$
	|	$\exists x.P$	Existential quantification		|	$E \mapsto E_1, E_2$		|	42 $\quad Int$
	|	$\texttt{emp}$	Empty heap					|	$\texttt{nil}$ $\quad nil$
	|	$P * Q$	Spatial conjunction					|	a $\quad Atom$
	|	$P -\!\!* Q$	Spatial implication						

2.3.1 Semantics : $s, h \models P$

Definitions

$$\begin{array}{rcl} h \sharp h' & : & dom(h) \text{ and } dom(h') \text{ are disjoint} \\ h \cdot h' & : & \text{the union of } h \text{ and } h' \text{ with disjoint domains} \\ P \models Q \quad \text{iff} & & \forall s, h \text{ if } s, h \models P \text{ then } s, h \models Q \end{array}$$

Atomic formulas

$$\begin{array}{rcccl} s, h & \models & E = E' & \text{iff} & [\![E]\!]s = [\![E']\!]s \\ s, h & \models & E \mapsto (E_1, E_2) & \text{iff} & dom(h) = \{[\![E]\!]s\} \text{ and } h([\![E]\!]s) = \langle [\![E_1]\!]s, [\![E_2]\!]s \rangle \end{array}$$

$E \mapsto (E_1, E_2)$ says that the heap has exactly one *location that looks like $E \mapsto (E_1, E_2)$.*

Classical formulas

$$\begin{array}{rcccl} s, h & \models & \texttt{false} & & never \\ s, h & \models & P \Rightarrow Q & \text{iff} & If\ s, h \models P\ Then\ s, h \models Q \\ s, h & \models & \exists x.P & \text{iff} & \exists v \in Val.[s|x \mapsto v], h \models P \end{array}$$

$\exists x.P$ says that we can assign a value to x in the stack to satisfy P.

Spatial formulas

$$\begin{array}{rcccl} s, h & \models & \texttt{emp} & \text{iff} & h = [] : \text{ empty heap} \\ s, h & \models & P * Q & \text{iff} & \exists h_0, h_1\ h_0 \sharp h_1,\ h = h_0 \cdot h_1 s, h_0 \models P \text{ and } s, h_1 \models Q \\ s, h & \models & P \mathrel{-\!\!*} Q & \text{iff} & \forall h', \text{If } h \sharp h' \text{ and } s, h' \models P \text{ Then } s, h \cdot h' \models Q \end{array}$$

$P * Q$ says that we can split the heap in two disjoint pieces, one that satisfies P and the other satisfies Q.

$P \mathrel{-\!\!*} Q$ says that for any piece of heap that satisfies P and is disjoint from the current heap, we can add it to the current heap and we will satisfies Q.

Notice that if with the current stack, there is no heap that makes P holding, $P \mathrel{-\!\!*} Q$ holds.

For readability of the formulas above, we omitted to write there that it is required that $FV(P) \subseteq dom(s)$ for $s, h \models P$, where $FV(P)$ is the set of free variables in P.

Extension of the syntax We can define various other connectives :

$$\begin{array}{rcl} \neg P & \equiv & P \Rightarrow \texttt{false} \\ \texttt{true} & \equiv & \neg(\texttt{false}) \\ P \vee Q & \equiv & (\neg P) \Rightarrow Q \\ P \wedge Q & \equiv & \neg(\neg P \vee \neg Q) \\ \forall x.P & \equiv & \neg(\exists x.\neg P) \\ E \hookrightarrow a, b & \equiv & \texttt{true} * (E \mapsto a, b) \end{array}$$

$E \hookrightarrow a, b$ says that the heap has among others one location that looks like $E \mapsto a, b$.

2.3.2 Deduction rules for $\models$

Definition

$$P \models Q \quad \text{iff} \quad \forall s, h.\ s, h \models P \text{ implies } s, h \models Q$$

- The usual rules of classical logic are sound for $\models$
- $*$ is commutative, associative, with unit `emp`
- $\frac{P' \models P \quad Q' \models Q}{P' * Q' \models P * Q}$

- $\frac{R*P \models Q}{R \models P \twoheadrightarrow Q}$
- $\frac{R \models P \twoheadrightarrow Q \quad R' \models P}{R*R' \models Q}$
- No weakening[2] : $P * Q \not\models P$
- No contraction[3] : $P \not\models P * P$

2.4 Triples

2.4.1 Interpretation

$\{P\}C\{Q\}$ true iff $\forall s, h$ If $s, h \models P$ and $FV(Q) \subseteq dom(s)$ Then
- C, s, h is safe
- if $C, s, h \rightsquigarrow^* s', h'$ then $s', h' \models Q$

Notice that this interpretation of triple is different from the usual Hoare triple's interpretation since $\{P\}C\{Q\}$ true implies that C can be executed from any state satisfying P.

The usual Hoare triple definition would be :

$\{P\}C\{Q\}$ true iff $\forall s, h$ If
- $s, h \models P$ and $FV(Q, C) \subseteq dom(s)$
- C, s, h is safe

Then
- if $C, s, h \rightsquigarrow^* s', h'$ then $s', h' \models Q$

2.4.2 Utility of triples for the analysis

From the interpretation of triples we have that :

√ If we know that $\{P\}C\{\texttt{true}\}$ is true then
we will know that C is safe to execute in any state satisfying P.

√ If we know that $\{P\}C\{Q\}$ is true then
we will know that C is safe to execute in any state satisfying P
and that from those states any terminal state satisfies Q.

This is the main point of our analysis. So we will define some rules that prove that a triple is true and use them for the analysis.

In the chapter 3, we will give for each statement a rule of the form $\{wlp(P, C)\}C\{P\}$ and the proofs that for each statement those triplets are true.

In the chapter 4, we will give for each statement a rule of the form $\{P\}C\{sp(P, C)\}$ and the proofs that for each statement, those triplets are true.

2.4.3 Axioms

Sequencing

$$\frac{\{P\}C\{Q\} \quad \{Q\}C'\{R\}}{\{P\}C; C'\{R\}}$$

Consequence

$$\frac{P \models P' \quad \{P'\}C\{R'\} \quad R' \models R}{\{P\}C\{R\}}$$

[2] because we have a notion of size of the heap with $*$
[3] as well

Frame Axiom Introduction

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \qquad \mathit{ModifiesOnly}(C) \cap FV(R) = \emptyset$$

$\mathit{ModifiesOnly}(C)$: set of variables appearing to the left of := in C and not dereferenced.

To know why this restriction for the Frame Axiom is necessary and sufficient, see Appendix A.

We can do local reasoning since because of the Frame Axiom, a specification can concentrate on only those cells that a program accesses.

2.5 Complements

- Computability : deciding the validity of an assertion is not recursively enumerable but if quantifiers are prohibited, the validity of an assertion is algorithmically decidable. (Yang and Calcagno [1])
- There exists a new version of BI [6, 7], to permit unrestricted address arithmetic, all values are integers (included addresses).
- There exists a intuitionist version of BI. A property is satisfied in a memory if it is for any extension of the heap. An inconvenient is that the property `emp` can never be satisfied.

Chapter 3

Backward analysis

3.1 Introduction

In the paper [4], the authors give axioms of the form $\{wlp(P, C)\}C\{P\}$ for some commands C and prove the correctness of the axioms for the commands $E.i := E'$ and $x := \texttt{cons}(E_1, E_2)$.

We have defined the wlp for the other commands and prove the correctness of all the wlp.

3.2 Definitions of the wlp

wlp stands for weakest liberal precondition where"liberal" means that we collect the configurations with infinite computation.

3.2.1 Preliminary definition

If we have that $P\{E/x\}$ is the formula P where x has been syntactically replaced by the formula E
we define

$$s, h \models P[E/x] \quad \text{iff} \quad s[x \mapsto [\![E]\!]s], h \models P$$

We define this because if $s, h \models P[E/x]$, we have :

- $[\![E]\!]s$ is defined, so the command $x := E$ can be executed
- $s[x \mapsto [\![E]\!]s], h \models P$, so P holds after the execution of $x := E$

So we will use this for the precondition of the assignment $x := E$.

This definition might look like an add to the language BI, but it is not since in Appendix E, we prove that $P[E/x] \equiv P\{E/x\} \wedge is(E)$,
where $is(E) \equiv (E = E)$ means that E has a value, we have $s, h \models E = E$ iff $[\![E]\!]s$ is defined and $P \equiv Q$ iff $P \models Q \wedge Q \models P$.

If $FV(E) \subseteq FV(P)$ then, $P[E/x] \equiv P\{E/x\}$, in fact we only need $FV(E) \subseteq FV(P\{E/x\})$.

The use of the distinction between [/] and {/}, can be view in the example : $\{\texttt{true}\}x := y\{\texttt{true}\}$ is false but $\{y = y\}x := y\{\texttt{true}\}$ is true.

Remark : in other version of BI [7], there is no need of distinction between $P[E/x]$ and $P\{E/x\}$, since the triples needs $FV(C, Q) \subseteq dom(s)$ to be true and not only $FV(Q) \subseteq dom(s)$ like here.

3.2.2 wlp

$$
\begin{array}{lcl}
wlp(P,\ x := E) & = & P[E/x] \\
wlp(P,\ x := E.i) & = & \exists x_1 \exists x_2.(P[x_i/x] \wedge (E \hookrightarrow x_1, x_2)) \\
 & & \quad with\ \ x_i \notin FV(E, P) \\
wlp(P,\ E.1 := E') & = & \exists x_1 \exists x_2.(E \mapsto x_1, x_2) * ((E \mapsto E', x_2) \mathrel{-\!\!*} P) \\
 & & \quad with\ \ x_i \notin FV(E, E', P) \\
wlp(P,\ E.2 := E') & = & \exists x_1 \exists x_2.(E \mapsto x_1, x_2) * ((E \mapsto x_1, E') \mathrel{-\!\!*} P) \\
 & & \quad with\ \ x_i \notin FV(E, E', P) \\
wlp(P,\ x := \texttt{cons}(E_1, E_2)) & = & \forall x'.(x' \mapsto E_1, E_2) \mathrel{-\!\!*} P[x'/x] \\
 & & \quad with\ \ x' \notin FV(E_1, E_2, P) \\
wlp(P,\ \texttt{dispose}(E)) & = & P * (\exists a \exists b.(E \mapsto a, b)) \\
 & & \quad with\ \ a, b \notin FV(E) \\
wlp(P,\ C_1; C_2) & = & wlp(wlp(P, C_2), C_1) \\
wlp(P,\ if\ \ E\ \ then\ \ C_1\ \ else\ \ C_2) & = & (E = \texttt{true} \wedge wlp(P, C_1)) \vee (E = \texttt{false} \wedge wlp(P, C_2)) \\
wlp(P,\ if\ \ E\ \ then\ \ C_1) & = & (E = \texttt{true} \wedge wlp(P, C_1)) \vee (E = \texttt{false} \wedge P) \\
wlp(P,\ while\ \ E\ \ do\ \ C_1) & = & gfp^{\models}_{\texttt{true}}\ \lambda X.((E = \texttt{true} \wedge wlp(X, C_1)) \\
 & & \qquad \vee (E = \texttt{false} \wedge P)) \\
 & & \\
wp(P,\ while\ \ E\ \ do\ \ C_1) & = & lfp^{\models}_{\texttt{false}}\ \lambda X.((E = \texttt{true} \wedge wp(X, C_1)) \vee P)
\end{array}
$$

We define $P \equiv Q$ iff $P \models Q$ and $Q \models P$.
Here we write $gfp^{\models}_{\texttt{true}}\ \lambda X.F(X)$ for a BI formula P which satisfies :

- $P \equiv F(P)$
- for any formula Q, ($Q \equiv F(Q)$ implies $Q \models P$)

This BI formula does not always exists and like any formula is not unique. To express the $wlp(P,\ while\ \ E\ \ do\ \ C_1)$ in any case we could take
$wlp(P, while\ \ E\ \ do\ \ C_1) = (E = \texttt{false} \wedge P)$.
We would have a complete and correct analysis, but in this case the *wlp* will not be the "weakest" pre condition.
We expressed the $wlp(P,\ while\ \ E\ \ do\ \ C_1)$ with a fixpoint to give a way to be able sometimes to compute a more precise precondition by iterations of applying the function $\lambda X.((E = \texttt{true} \wedge wlp(X, C_1)) \vee (E = \texttt{false} \wedge P))$ to formulas starting by `true` and stopping when two consecutive formulas are equivalent in the sens of $\equiv$.

There are some explanations of the formulas in the case when $P = \texttt{true}$ in 4.2.

Remark : If we would like to implement the analysis, for the *while* case, we would need a theorem prover[1]. Deciding whether $P \models Q$ is the same problem as deciding the validity of $P \Rightarrow Q$. As said in the previous chapter, this is not recursively enumerable, so we might not be able to compute the gfp. If we would like to implement the analysis, we could at least use the approximation $E = \texttt{false} \wedge P$.

3.3 Proofs of the wlp

First we should precise that in this chapter we only want to prove that our analysis is sound. That is we proof that all the triples $\{wlp(P, C)\}C\{P\}$ are true but we do not proof that the *wlp*'s formulas are actually the weakest one. This is the case in fact for the simple statements but not for the *while*.

Definition : $\gamma(P) = \{s, h \mid s, h \models P\}$

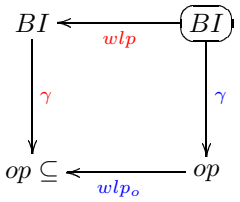
[1] there have been some works done for proof-search in BI Logic [3] but, to our knowledge, only for the intuitionist version of BI

Definition of wlp_o We define the wlp in the operational domain :
$wlp_o(\Delta, C) = \{s, h \mid C, s, h \text{ is safe} \wedge (\text{if } C, s, h \rightsquigarrow^* s', h' \text{ then } s', h' \in \Delta\}$

We can rewrite the definition of a true triple as :

$$\{wlp(P, C)\}C\{P\} \text{ true iff } (\gamma(wlp(P, C)) \cap \{s, h \mid FV(P) \subseteq dom(s)\}) \subseteq wlp_o(\gamma(P), C)$$

So we will prove that for each C, and P, we have $\gamma(wlp(P, C)) \subseteq wlp_o(\gamma(P), C)$



Here, $op = S \times H$

3.3.1 Proofs

We just need to express wlp_o for each command and we will prove the correctness by induction on the syntax of C. *(we wrote the proofs but did not type them)*

Chapter 4

Forward analysis

4.1 Introduction

In the chapter 3, we have given for each C and each P a $wlp(P, C)$ such that $\{wlp(P, C)\}C\{P\}$ is true.

We can not always define $sp(P, C)$, where sp stands for strongest post condition. That is we can find a C and a P such that there exists no Q that makes $\{P\}C\{Q\}$ true. This is so because to be true a triple asks for C to be executable from all states satisfying P *(and also such that* $FV(Q) \subseteq dom(s)$*)* which is obviously not the case for any C and P.
For example $\{\texttt{true}\}x = nil; y = x.1; \{?\}$ has no solution, since all states satisfy P but the command can never be executed*(*$\texttt{nil}.1$ *not defined).*

So we have to split the analysis into two steps. The first step is to check whether C is executable from all states satisfying P or not. The second step is to give $sp(P, C)$ that makes the triple $\{P\}C\{sp(P, C)\}$ true if C is executable from all states satisfying P.

4.2 Step 1 : $wlp(\texttt{true}, C)$

We have that C is executable from any state satisfying P iff $P \models wlp(\texttt{true}, C)$.
So for the first step we just need to express the $wlp(\texttt{true}, C)$'s formulas.

Definition $is(E) \equiv E = E$, it means that E has a value in the current memory.

$$
\begin{array}{lcl}
wlp(\texttt{true}, \quad x := E) & = & is(E) \\
wlp(\texttt{true}, \quad x := E.i) & = & \exists x_1 \exists x_2 . E \hookrightarrow x_1, x_2 \\
 & & \quad with \;\; x_i \notin FV(E) \\
wlp(\texttt{true}, \quad E.i := E') & = & \exists x_1 \exists x_2 . (E \mapsto x_1, x_2) \wedge is(E') \\
 & & \quad with \;\; x_i \notin FV(E, E') \\
wlp(\texttt{true}, \quad x := \texttt{cons}(E_1, E_2)) & = & is(E_1) \wedge is(E_2) \\
wlp(\texttt{true}, \quad \texttt{dispose}(E)) & = & \exists x_1 \exists x_2 . E \hookrightarrow x_1, x_2 \\
 & & \quad with \;\; x_i \notin FV(E) \\
wlp(\texttt{true}, \quad C_1; C_2) & = & wlp(wlp(\texttt{true}, C_2), C_1) \\
wlp(\texttt{true}, \quad if \;\; E \;\; then \;\; C_1 \;\; else \;\; C_2) & = & (E = \texttt{true} \wedge wlp(\texttt{true}, C_1)) \\
 & & \vee (E = \texttt{false} \wedge wlp(\texttt{true}, C_2)) \\
wlp(\texttt{true}, \quad if \;\; E \;\; then \;\; C_1) & = & (E = \texttt{true} \wedge wlp(\texttt{true}, C_1)) \vee E = \texttt{false} \\
wlp(\texttt{true}, \quad while \;\; E \;\; do \;\; C_1) & = & gfp^{\models}_{\texttt{true}} \;\; \lambda X . ((E = \texttt{true} \wedge wlp(X, C_1)) \\
 & & \quad \vee E = \texttt{false})
\end{array}
$$

Some explanation :

$wlp(\texttt{true}, x := E) \;=\; is(E)$
we need E to have a value to execute the command
$wlp(\texttt{true}, x := E.i) \;=\; (\exists x_1, x_2.\ E \hookrightarrow x_1, x_2)$
we need E to be a pointer assigned which point to two values
$wlp(\texttt{true}, E_1.i := E_2) \;=\; ((\exists x_1, x_2.\ E_1 \hookrightarrow x_1, x_2) \wedge is(E_2))$
we need E_1 to be a pointer assigned which point to two values and we need E_2 to have a value
$wlp(\texttt{true}, x := \texttt{cons}(E_1, E_2)) \;=\; (is(E_1) \wedge is(E_2))$
we need E_1 and E_2 to have a value to execute the command
$wlp(\texttt{true}, \texttt{dispose}(E)) \;=\; (\exists x_1, x_2.\ E \hookrightarrow x_1, x_2)$
we need E to be a pointer assigned which point to two values
$wlp(\texttt{true}, C_1; C_2) \;=\; wlp(wlp(\texttt{true}, C_2), C_1)$
here, it is not the composition of $wlp(\texttt{true}, \square)$
so we will not do composition of the first step

4.3 Step 2 : $sp(P, C)$ in case $P \models wlp(\texttt{true}, C)$

First we define

$$\begin{array}{rll}
s, h \models P[E/x] & \text{iff} & s[x \mapsto [\![E]\!]s], h \models P \\
s, h \models x = E[E_2/y] & \text{iff} & s(x) = [\![E]\!]s[y \mapsto [\![E_2]\!]s] \\
s, h \models x = E[E_2/y].i & \text{iff} & s(x) = \pi_i(h([\![E]\!]s[y \mapsto [\![E_2]\!]s])) \\
s, h \models x \mapsto E_1[E_0/y], E_2[E_0/y] & \text{iff} & dom(h) = \{s(x)\} \\
 & & \wedge h(s(x)) =< [\![E_1]\!]s[y \mapsto [\![E_0]\!]s], [\![E_2]\!]s[y \mapsto [\![E_0]\!]s] >
\end{array}$$

In Appendix E, we prove that those definitions correspond to the syntactic substitution with some modification like in chapter 3. So that those new definitions are covered by the syntax of BI. Actually, $x = E.i$ is not a BI's formula, but we write it as a shortcut for the formula $\exists x_1, x_2.\ (E \hookrightarrow x_1, x_2) \wedge (x = x_i)$.

$$\begin{array}{lcl}
sp(P,\ x := E) & = & \exists x'.\ P[x'/x] \wedge x = E[x'/x] \\
 & & \quad with\ \ x' \notin FV(E, P) \\
sp(P,\ x := E.i) & = & \exists x'.\ P[x'/x] \wedge x = (E[x'/x]).i \\
 & & \quad with\ \ x' \notin FV(E, P) \\
sp(P,\ E_1.1 := E_2) & = & \exists x_1 \exists x_2.(E_1 \mapsto E_2, x_2) * ((E_1 \mapsto x_1, x_2) \mathrel{-\!\!*} P) \\
 & & \quad with\ \ x_i \notin FV(E_1, E_2, P) \\
sp(P,\ E_1.2 := E_2) & = & \exists x_1 \exists x_2.(E_1 \mapsto x_1, E_2) * ((E_1 \mapsto x_1, x_2) \mathrel{-\!\!*} P) \\
 & & \quad with\ \ x_i \notin FV(E_1, E_2, P) \\
sp(P,\ x := \texttt{cons}(E_1, E_2)) & = & \exists x'.(P[x'/x] * (x \mapsto E_1[x'/x], E_2[x'/x])) \\
 & & \quad with\ \ x' \notin FV(E_1, E_2, P) \\
sp(P,\ \texttt{dispose}(E)) & = & \exists x_1, x_2.\ ((E \mapsto x_1, x_2) \mathrel{-\!\!*} P) \\
 & & \quad with\ \ x_1, x_2 \notin FV(E, P) \\
sp(P,\ C_1; C_2) & = & sp(sp(P, C_1), C_2) \\
sp(P,\ if\ E\ then\ C_1\ else\ C_2) & = & sp(P \wedge E = \texttt{true}, C_1) \\
 & & \vee sp(P \wedge E = \texttt{false}, C_2) \\
sp(P,\ if\ E\ then\ C_1) & = & sp(P \wedge E = \texttt{true}, C_1) \\
 & & \vee (P \wedge E = \texttt{false}) \\
sp(P,\ while\ E\ do\ C_1) & = & (lfp^{\models}_{P}\ \lambda X.sp(X \wedge E = \texttt{true}, C_1) \vee X) \\
 & & \wedge (E = \texttt{false})
\end{array}$$

The lfp does not necessary exists (for example if the program enumerates the prime numbers, the formula would be infinite and BI does not have infinite $\vee$ or $\wedge$).

We could, if the lfp does not exist, have

$$sp(P,\ \ while\ E\ do\ C_1) \;=\; E = \texttt{false}$$

but in this case sp will not be the "strongest" post condition.

As discussed for the wlp case, if we would like to implement this analysis, we would not necessarily be able to compute the lfp because of the quantifiers. But we have that deciding the validity of an assertion without quantifiers is algorithmically decidable. So a solution for implementation could be to lose the quantifiers and lose some precision. For example, in the case of $x := E$, if x does not occur in E, we could replace $\exists x'.\, P[x'/x] \wedge x = E[x'/x]$ by $x = E$ and we could possibly keep some information from P that does not depend on x. Then we would no more have the sp but still have true triples. We did not look at the implementation problem, a solution for the $while$ could be to give $E = \texttt{false}$ when we can not compute the lfp.

Some explanation :

$sp(P,\ x := E) \ =\ \exists x'.\, P[x'/x] \wedge x = E[x'/x]$
with $x' \notin FV(E, P)$
x' plays the role of the previous value of x

$sp(P,\ x := E.i) \ =\ \exists x'.\, P[x'/x] \wedge x = (E[x'/x]).i$
with $x' \notin FV(E, P)$
x' plays the role of the previous value of x

$sp(P,\ E_1.1 := E_2) \ =\ \exists x_1 \exists x_2.(E_1 \mapsto E_2, x_2) * ((E_1 \mapsto x_1, x_2) \mathrel{-\!\!*} P)$
with $x_i \notin FV(E, E', P)$
it means that there were two previous values x_1 and x_2 such that if we take out E and add this previous E, P holds and actually E is pointing the updated cells

$sp(P,\ x := \texttt{cons}(E_1, E_2)) \ =\ \exists x'.(P[x'/x] * (x \mapsto E_1[x'/x], E_2[x'/x]))$
with $x' \notin FV(E_1, E_2, P)$
x' plays the role of the previous value of x we can split the memory into two parts, one part was holding P with the previous value of x, the other part is x that points to what it should

$sp(P,\ \texttt{dispose}(E)) \ =\ \exists x_1, x_2.\ ((E \mapsto x_1, x_2) \mathrel{-\!\!*} P)$
with $x_1, x_2 \notin FV(E, P)$
it means that there were two previous values, that E was pointing to, and if we add to the actual memory, E pointing to those values, we hold P as before the `dispose`

4.4 if $P \not\models wlp(\texttt{true}, C)$

If $P \not\models wlp(\texttt{true}, C)$, we can conclude that C can not be executable from all states satisfying P and for those from which it is executable, the final states satisfy $sp(P \wedge wlp(\texttt{true}, C), C)$.

Remark $P \not\models wlp(\texttt{true}, C)$ does not imply that the previous sp are necessary false.

For example, $\texttt{true} \not\models is(y)$ but $\{\texttt{true}\}x := y\{\exists x'.\ x = y\}$ is true, since we have the special restriction that $x := y$ as to be executable from all state satisfying P which is not true here but also that have $FV(\exists x'.\ x = y) \subseteq dom(s)$ which is $is(x) \wedge is(y)$ and implies that those states $\models is(y)$ and so the triple is true.

But this is a particular case, and we do not have $P \wedge$ "$FV(sp(P, C)) \subseteq dom(s)'' \models wlp(\texttt{true}, C)$.

This problem does not come only from the composition or if and $while$. For example,
$wlp(\texttt{true}, dispose(x)) = \exists x_1, x_2.\ x \hookrightarrow x_1, x_2$,
$sp(\texttt{true}, dispose(x)) = \exists x_1, x_2.\ (x \mapsto x_1, x_2) \mathrel{-\!\!*} \texttt{true}$
and "$FV(sp(\texttt{true}, dispose(x))) \subseteq dom(s)'' = is(x)$

but $\texttt{true} \wedge is(x) \not\models \exists x_1, x_2.\ x \hookrightarrow x_1, x_2$.

4.5 Proof of the sp

First we should precise that in this chapter we only want to prove that our analysis is sound. That is we do not prove that the *sp*'s formulas are actually the stongest one. This is the case in fact for the simple statements but not for the *while*.

We want to prove that :

$$\text{If } P \models wlp(\texttt{true}, C) \text{ then } \{P\}C\{sp(P, C)\} \text{ true}$$

Definition of sp_o We define the sp in the operational domain :
$sp_o(\Delta, C) = \{s', h' \mid \exists s, h \in \Delta.\ C, s, h \rightsquigarrow^* s', h'\}$

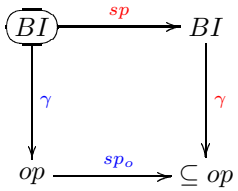
So we can rewrite the definition of a true triple as :

$$\{P\}C\{Q\} \text{ true iff } P \models wlp(\texttt{true}, C) \wedge sp_o(\gamma(P), C) \subseteq \gamma(Q)$$

So we want to prove for each command C that :

$$\text{If } P \models wlp(\texttt{true}, C) \text{ then } sp_o(\gamma(P), C) \subseteq \gamma(sp(P, C))$$

.



But since sp_o is defined such that it only collect the final states of successful computation, we only have to prove that for each command C :

$$sp_o(\gamma(P), C) \subseteq \gamma(sp(P, C))$$

See appendix B.

4.6 Add of an error state

There is a second approach, the one we first took, that is to add an error state to our domain. So we make the transition function a total function. We have changed the syntax of BI and the interpretation of a triple. We now produce for all P and C, a triple $\{P\}C\{Q\}$ which is true. For our analysis, to know that a program is executed without error from the states satisfying P, we will just have to check if the error state satisfies Q.

4.6.1 Configurations

Domain : (stack S + heap H) or Error The domain is the union of the previous domain with $\{\Omega_o\}$.

Ω_o is the error memory in the operational domain.

4.6.2 Operational semantics: $C, m \rightarrow m'$ $C, m \rightarrow C', m'$

$$\begin{array}{rcl} C, m \rightarrow m' & \text{iff} & (m = \Omega_o \wedge m' = \Omega_o) \\ & & \vee (m = s, h \wedge m' = \Omega_o \wedge C, s, h \not\rightsquigarrow) \\ & & \vee (m = s, h \wedge m' = s', h' \wedge C, s, h \rightsquigarrow s', h') \\ C, m \rightarrow C', m' & \text{iff} & m = s, h \wedge m' = s', h' \wedge C, s, h \rightsquigarrow C', s', h' \end{array}$$

Then the operational semantics $\rightarrow$ is a total function.
*There are no more **stuck** configurations.*

4.6.3 BI^e

Syntax of BI^e

P	::=	Q	BI's formulas
	|	`Err`	the atomic formula for error
	|	`true`	added since the law of the excluded middle does not hold anymore
	|	$\forall x.P$	as well

We add to the syntax of BI a predicate `Err` *that holds only for the state Ω_o.*

Semantics: $m \models^e P$

m	$\models^e$	P	iff	$(m = \Omega_o \wedge \Omega_o \models^e P)$
				$\vee (m = s, h \wedge s, h \models^e P)$
Ω_o	$\models^e$	`Err`		*always*
Ω_o	$\models^e$	$P \Rightarrow Q$	iff	$If\ \Omega_o \models^e P\ Then\ \Omega_o \models^e Q$
Ω_o	$\models^e$	`true`		*never*
Ω_o	$\models^e$	$\forall x.P$		*never*
Ω_o	$\models^e$	*one of the others BI's formulas*		*never*
s, h	$\models^e$	`Err`		*never*
s, h	$\models^e$	`true`		*always*
s, h	$\models^e$	$\forall x.P$	iff	$\forall v \in Val.[s\|x \mapsto v], h \models P$
s, h	$\models^e$	*P a BI's formulas*	iff	$s, h \models P$

All the special BI's formulas define a "normal" memory, so Ω_o does not satisfy any of them. The only predicate that Ω_o satisfies is `Err`*. The usual meaning of $\Rightarrow$ is valid for error or non error memory. $\exists x...$ had the meaning that we could assign a value to x in the stack, so it supposed that the stack exist, and so Ω_o does not satisfy any $\exists$ or $\forall$.*

4.6.4 Triples

The previous interpretation of triples can be written

$$\{P\}C\{Q\} \text{ iff } P \models wlp(\texttt{true}, C) \wedge sp_o(\gamma(P), C) \subseteq \gamma(Q)$$

.

Remember that was

- C is executable from all states satisfying P
- all final states of a computation of C from P satisfy Q

Now we have

$$\{P\}C\{Q\} \text{ iff } sp_o^e(\gamma^e(P), C) \subseteq \gamma^e(Q)$$

with $\gamma^e(P) = \{m \mid m \models^e P\}$.

Which is

- if Ω_o satisfies P then Ω_o satisfies Q
- if C can not be executable from all "normal" states satisfying P, then Ω_o satisfies Q
- all "normal" final states of a computation of C from P satisfy Q

We could express sp^e with the two steps of the previous analysis :
Step 1 : check if $P \models wlp(\texttt{true}, C)$
Step 2 : if yes then $sp^e(P, C) = sp(P, C)$
if not then $sp^e(P, C) = Err \vee sp(P \wedge wlp(\texttt{true}, C), C)$.

Notice that we do not have $sp^e(P, C_1; C_2) = sp^e(sp^e(P, C_1), C_2)$ in the above definition since $sp : BI \rightarrow BI^e$ and so also $sp^e : BI \rightarrow BI^e$ but we can extend the definition of sp^e so that $sp^e(P, C_1; C_2) = sp^e(sp^e(P, C_1), C_2)$ since $\Omega_0 \not\models P \wedge wlp(\texttt{true}, C)$.

4.6.5 BI^2

Introduction

If we do not want sp^e written with two steps. And if we want that $sp^e(P, C_1; C_2) = sp^e(sp^e(P, C_1), C_2)$. This is less efficient so this section does not need to be read.

We said that we have

$$\{P\}C\{Q\} \text{ iff } sp^e_o(\gamma^e(P), C) \subseteq \gamma^e(Q)$$

with $\gamma^e(P) = \{m \mid m \models^e P\}$.

Which is

- if Ω_o satisfies P then Ω_o satisfies Q
- if C can not be executable from all "normal" states satisfying P, then Ω_o satisfies Q
- all "normal" final states of a computation of C from P satisfy Q

We can rewrite this as

- $\texttt{Err} \models^e P$ then $\texttt{Err} \models^e Q$
- if $P \not\models^e wlp(\texttt{true}, C)$ then $\texttt{Err} \models^e Q$
- $sp(P \wedge wlp(\texttt{true}, C), C) \models Q$, *here we write sp and not sp^e because we are in the case where we go from "normal states" to "normal states"*

Here we want to find an sp^e such that $\{P\}C\{sp^e(P, C)\}$ is true

so we would like sp^e to be like

$$\begin{array}{rcl} sp^e(P, C) & = & (\texttt{Err} \wedge P) \\ & & \vee(\texttt{Err} \wedge (P \not\models^e wlp(\texttt{true}, C))) \\ & & \vee sp(P \wedge wlp(\texttt{true}, C), C) \end{array}$$

but $\not\models$ can not be expressed in BI^e,
so we have extended its syntax into BI^2 with $\forall^2$ and $\exists^2$ to get :

$$\begin{array}{lll} \texttt{true} \models^e P & \text{iff} & m \models^2 \forall^2 m'.\ P \\ P \models^e Q & \text{iff} & m \models^2 \forall^2 m'.\ P \Rightarrow Q \end{array}$$

and we can express :

$$\begin{array}{rcl} sp^2(P, C) & = & (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \neg wlp(\texttt{true}, C))))) \\ & & \vee sp(P \wedge wlp(\texttt{true}, C), C) \end{array}$$

meaning that we have the state error if the state error satisfies P or if there is a state satisfying P from which C is not executable, and we also have all "normal" post states.

The idea of the necessity of BI^2 is that we have to express that Ω_o satisfies a formula which says that there was a previous "normal" memory from which the command can not be executed. This previous "normal" memory can not be expressed from Ω_o, so it can not be expressed in BI^e.

Syntax of BI^2

$$\begin{array}{rcll} P & ::= & Q & BI^e\text{'s formulas} \\ & | & \forall^e m.\ P & \\ & | & \exists^e m.\ P & \end{array}$$

Semantics: $m \models^2 P$

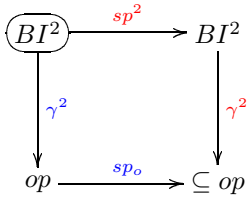
$$\begin{array}{llll} m' \models^2 \forall^2 m.\ P & \text{iff} & \forall m.\ (m = s, h \Rightarrow m \models^2 P) \\ m' \models^2 \exists^2 m.\ P & \text{iff} & \exists m.\ (m = s, h \wedge m \models^2 P) \\ m' \models^2 other\ P & \text{iff} & same\ definition\ as\ for\ BI^e \end{array}$$

Triples

We have $sp^2_o(\Delta, C) = \{m' \mid \exists m \in \Delta.\ C, m \rightarrow^* m'\}$

and then

$$\{P\}C\{Q\} \text{ true iff } sp^2_o(\gamma^2(P), C) \subseteq \gamma^2(Q)$$



Preliminary definition

First we define

$$\begin{array}{rcl} s, h \models P[E/x] & \text{iff} & \exists m'.\ m' = s', h' \wedge m' \models P \wedge h' = h \wedge s'(x) = [\![E]\!]s \wedge \exists v\ .\ s = s'[x \mapsto v] \\ s, h \models x = E[E_2/y] & \text{iff} & \exists m'.\ m' = s', h' \wedge h' = h \wedge s(x) = [\![E]\!]s' \wedge \exists v\ .\ (s' = s[y \mapsto v] \wedge v = [\![E_2]\!]s) \\ s, h \models x = E[E_2/y].i & \text{iff} & \exists m'.\ m' = s', h' \wedge h' = h \wedge s(x) = \pi_i(h'([\![E]\!]s')) \\ & & \wedge \exists v\ .\ (s' = s[y \mapsto v] \wedge v = [\![E_2]\!]s) \\ s, h \models x \mapsto E_1[E_0/y], E_2[E_0/y] & \text{iff} & \exists m'.\ m' = s', h' \wedge h' = h \wedge dom(h) = \{s(x)\} \wedge h(s(x)) =< [\![E_1]\!]s', [\![E_2]\!]s' > \\ & & \wedge \exists v\ .\ (s' = s[y \mapsto v] \wedge v = [\![E_0]\!]s) \end{array}$$

We should prove that those definitions correspond to the syntactic substitution with some modification like in chapter 3. So that those new definitions are covered by the syntax of BI^2. We should look more carefully about $\Omega_o \models^2$ ...[.../...], we did not looked at this right now since it is not directly useful in the proofs we made.

Definitions of the sp^2

$$\begin{array}{rcl} sp^2(P, C) & = & (\mathtt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \neg wlp(\mathtt{true}, C))))) \\ & & \vee sp(P \wedge wlp(\mathtt{true}, C), C) \end{array}$$

For ; and *if* and *while*, the formulas are not written like that. If the formulas where like that, this would do the same work as the method with the two steps, supposing that for composition, we do the first step only for the composition and not for the subcommand separately. For the composition, we have written $sp^2(P, C_1; C_2) = sp^2(sp^2(P, C_1), C_2)$, which implies that we do more

"first step" work than in the formula above.
Remark : this formula above is just an explanation of the formulas written after since the sp written is not the $sp : BI \to BI$ since $P \wedge wlp(\texttt{true}, C)$ is in BI^2 but $\Omega_o \not\models P \wedge wlp(\texttt{true}, C)$.

$$
\begin{array}{lcl}
sp^2(P, & x := E) & = (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \neg(E = E))))) \\
& & \vee(\exists x'.\ P[x'/x] \wedge x = E[x'/x]) \\
sp^2(P, & x := E.i) & = (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \\
& & \quad \neg(\exists x_1, x_2.\ E \hookrightarrow x_1, x_2)))) \\
& & \vee(\exists x'.\ P[x'/x] \wedge x = (E[x'/x]).i) \\
& & \quad with\ \ x' \notin FV(E, P) \\
sp^2(P, & E_1.1 := E_2) & = (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \\
& & \quad (\neg(\exists x_1, x_2.\ E_1 \hookrightarrow x_1, x_2) \vee \neg(E_2 = E_2)))))) \\
& & \vee(\exists x_1 \exists x_2.(E_1 \mapsto E_2, x_2) * ((E_1 \mapsto x_1, x_2) \mathrel{-\!\!*} P)) \\
& & \quad with\ \ x_i \notin FV(E, E', P) \\
sp^2(P, & E_1.2 := E_2) & = (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \\
& & \quad (\neg(\exists x_1, x_2.\ E_1 \hookrightarrow x_1, x_2) \vee \neg(E_2 = E_2)))))) \\
& & \vee(\exists x_1 \exists x_2.(E_1 \mapsto x_1, E_2) * ((E_1 \mapsto x_1, x_2) \mathrel{-\!\!*} P)) \\
& & \quad with\ \ x_i \notin FV(E, E', P) \\
sp^2(P, & x := \texttt{cons}(E_1, E_2)) & = (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \\
& & \quad (\neg(E_1 = E_1) \vee \neg(E_2 = E_2)))))) \\
& & \vee(\exists x'.(P[x'/x] * (x \mapsto E_1[x'/x], E_2[x'/x]))) \\
& & \quad with\ \ x' \notin FV(E_1, E_2, P) \\
sp^2(P, & \texttt{dispose}(E)) & = (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \\
& & \quad \neg(\exists x_1, x_2.\ E \hookrightarrow x_1, x_2))))) \\
& & \vee(\exists x_1, x_2.\ ((E \mapsto x_1, x_2) \mathrel{-\!\!*} P)) \\
& & \quad with\ \ x_1, x_2 \notin FV(E, P) \\
sp^2(P, & C_1; C_2) & = sp^2(sp^2(P, C_1), C_2) \\
sp^2(P, & if\ \ E\ \ then\ \ C_1\ \ else\ \ C_2) & = (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \neg(E = E))))) \\
& & \vee(sp^2(P \wedge E = \texttt{true}, C_1)) \\
& & \vee(sp^2(P \wedge E = \texttt{false}, C_2)) \\
sp^2(P, & if\ \ E\ \ then\ \ C_1) & = (\texttt{Err} \wedge (P \vee (\exists^2 m.\ (P \wedge \neg(E = E))))) \\
& & \vee(sp^2(P \wedge E = \texttt{true}, C_1)) \\
& & \vee(P \wedge E = \texttt{false}) \\
sp^2(P, & while\ \ E\ \ do\ \ C_1) & = (lfp\ ^{\models^2_P}\ \lambda X. \\
& & \quad (\texttt{Err} \wedge (X \vee (\exists^2 m.\ (X \wedge \neg(E = E))))) \\
& & \quad \vee(sp^2(X \wedge E = \texttt{true}, C_1)) \\
& & \quad \vee X) \\
& & \wedge (E = \texttt{false} \vee \texttt{Err})
\end{array}
$$

The lfp does not necessary exists as seen before.
To have a total sp^2 function, we could, if the lfp does not exist, have

$$sp^2(P, \quad while\ \ E\ \ do\ \ C_1) \quad = \quad E = \texttt{false} \vee \texttt{Err}$$

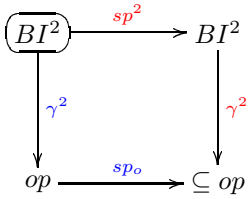
As discussed before, if we would like to implement this analysis, we would not necessarily be able to compute the lfp because of the quantifiers. A solution for the *while* could be to give $E = \texttt{false} \vee \texttt{Err}$ when we can compute. We would not have a sp but still have true triples.

Proof of the sp^2
We have

$$\{P\}C\{Q\} \text{ true iff } \ sp^2_o(\gamma^2(P), C) \subseteq \gamma^2(Q)$$

So we want :

$$sp^2_o(\gamma^2(P), C) \subseteq \gamma^2(sp^2(P, C))$$



We just need to express sp_o^2 for each command and we will proof the correctness by induction on the syntax of C. It is almost the same proofs as for the sp's.

See Appendix C.

Chapter 5

Partitioning

5.1 Introduction

In the previous chapter, we have seen a backward and a forward analyzes that could be run on a program using BI logic.

In this chapter we are interested in using BI as an interface language for modularity with other analyzes.
We could give with the analyzes of the previous chapters some properties that holds before and

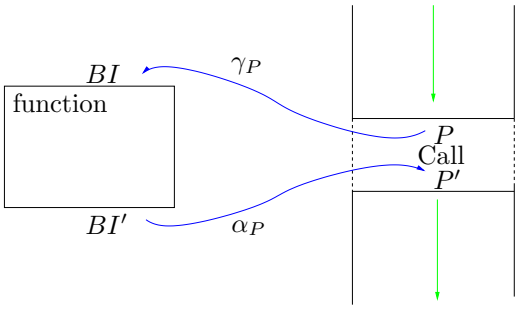


Figure 5.1: Use BI's analysis in a analysis over a domain P

after a piece of program *(BI and BI' in fig 5.1)*. And then for an other analysis over a domain P, if we give a translation from P into BI's formulas *(γ_P in fig)* and from BI's formulas into P *(α_P in fig)*, we would be able to use the result of BI's analysis.

As an example, we have chosen an analysis from chapter *4.2* of [2]. It is an analysis that says if some pointers can not for sure transitively reach a same location.

In fact this analysis gives a partition that is a set of collections of pointers that may reach a same location. Since the set of variables of a program is finite, we have represented the partition as a set of couple of variables that for sure do not reach each other. If in the paper [1] they give $\{V, W/X, Y, Z\}$ as a partition for a program with variables in $\{V, W, X, Y, Z, A\}$, we will have for our partition $\{[V\backslash X], [V\backslash Y], [V\backslash Z], [V\backslash A], [W\backslash X], [W\backslash Y], [W\backslash Z], [W\backslash A], [A\backslash X], [A\backslash Y], [A\backslash Z]\}$ (we naturally have $[a\backslash b] \in P \Rightarrow [b\backslash a] \in P$).

[1] We describe the partitions domain of the paper as a footnote since we do not use it anymore after this example. $\{V, W/X, Y, Z\}$ means that :
- there *may* be a way to "link" V and W
- there is definitelly no way to "link" V or W to any other variable, even if they are not in $\{X, Y, Z\}$
- there *may* be a way to "link" X and Y, X and Z or Y and Z
- there are definitely no way to "link" them to any other variable

We also have restricted the language over which the analysis is done to match with our work on BI.

In this chapter, we describe the analysis over the partitions, then we give the translation from Partitions to BI's formula, and then for some commands, we prove that the sp_P's formulas are right.

5.2 Partitions analysis

5.2.1 Commands

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
E & ::= & x := \texttt{nil} & & & B & ::= & x = \texttt{nil} & C & ::= & E \\
 & | & x := y & & & & | & x = y & & | & \texttt{new}(x) \quad as\ \text{“}x := \texttt{cons}(\texttt{nil},\texttt{nil})\text{”} \\
 & | & x := y.i & i & ::= & 1|2 & | & x = y.i & & | & C_1; C_2 \\
 & | & x.i := y & & & & | & x.i = y & & | & if\ \ B'\ \ then\ \ C_1\ \ else\ \ C_2 \\
 & & & & & & B' & ::= & B & | & if\ \ B'\ \ then\ \ C_1 \\
 & & & & & & & | & \neg\, B & | & while\ \ B'\ \ do\ \ C_1
\end{array}
$$

We do not write the operational semantics of the above commands, they are the same as in the previous chapters.

5.2.2 Partition definition

A partition is a set of $[a\backslash b]$. $[a\backslash b]$ means that a and b are not pointers that can reach each other or can be both reach by a same location. If $[a\backslash b]$ is not in the partition, a and b may reach each other. If $[a\backslash b] \in P$ then $[b\backslash a] \in P$.

5.2.3 Strongest Post Conditions

We have $sp_P : Partitions \times Commands \rightarrow Partitions$.
Were $Partions$ is the power set of $\{[a\backslash b] \mid a, b \in Var\}$ ordered by the inclusion of sets.

The analysis will start with the top partition, $\forall a, b\ a \neq b\ .\ [a\backslash b] \in P$.
Then we express the postconditions for our commands.

$$
\begin{array}{rrcl}
sp_P(P, & \texttt{new}(x)) & = & P \setminus \{[x\backslash z] \mid \forall z\} \cup \{[x\backslash z] \mid \forall z\ z \neq x\} \\
sp_P(P, & x := \texttt{nil}) & = & P \setminus \{[x\backslash z] \mid \forall z\} \cup \{[x\backslash z] \mid \forall z\} \\
sp_P(P, & x = \texttt{nil}) & = & sp_P(P, x := \texttt{nil}) \\
sp_P(P, & x := y) & = & P \setminus \{[x\backslash z] \mid \forall z\} \cup \{[x\backslash z] \mid [y\backslash z] \in P\} \\
sp_P(P, & x = y) & = & sp_P(P, x.i := y) \\
sp_P(P, & x := y.i) & = & P \setminus \{[x\backslash z] \mid \forall z\} \cup \{[x\backslash z] \mid [y\backslash z] \in P,\ z \neq x\} \\
sp_P(P, & x = y.i) & = & sp_P(P, x.i := y) \\
sp_P(P, & x.i := y) & = & P \setminus \{[v\backslash w] \mid ([y\backslash w] \notin P \wedge [x\backslash v] \notin P) \\
 & & & \qquad \vee([x\backslash w] \notin P \wedge [y\backslash v] \notin P)\} \\
sp_P(P, & x.i = y) & = & sp_P(P, x.i := y) \\
sp_P(P, & C_1; C_2) & = & sp_P(sp_P(P, C_1), C_2) \\
sp_P(P, & if\ \ B\ \ then\ \ C_1\ \ else\ \ C_2) & = & sp_P(sp_P(P, B), C_1) \cap sp_P(P, C_2) \\
sp_P(P, & if\ \ \neg\, B\ \ then\ \ C_1\ \ else\ \ C_2) & = & sp_P(P, C_1) \cap sp_P(sp_P(P, B), C_2) \\
sp_P(P, & if\ \ B\ \ then\ \ C_1) & = & sp_P(sp_P(P, B), C_1) \cap P \\
sp_P(P, & if\ \ \neg\, B\ \ then\ \ C_1) & = & sp_P(P, C_1) \cap sp_P(P, B) \\
sp_P(P, & while\ \ B\ \ do\ \ C_1) & = & gfp^{\subseteq}_{P}\ \lambda X.\ sp_P(sp_P(X, B), C_1) \cap X \\
sp_P(P, & while\ \ \neg\, B\ \ do\ \ C_1) & = & gfp^{\subseteq}_{P}\ \lambda X.\ sp_P(X, C_1) \cap sp_P(X, B) \\
sp_P(P, & \neg\, B)) & = & P
\end{array}
$$

Here we give some explaination why the case of $new(x)$ is not the same as the case of $x := \texttt{nil}$. It's just that in our definition of $[a\backslash b]$, we consider that if x does not point to a location (i.e. x

is `nil` or is a number) then $[x\backslash x]$ holds but if x points to a location, which is the case after a allocation $new(x)$, then $[x\backslash x]$ does not hold.
This can be view in section 5.3.

5.2.4 Weakest Pre Conditions

$$
\begin{array}{rrcl}
wlp_P(P, & B) & = & sp_P(P, B) \\
wlp_P(P, & \texttt{new}(x)) & = & P \setminus \{[x\backslash z] \mid \forall z\} \\
wlp_P(P, & x := \texttt{nil}) & = & P \setminus \{[x\backslash z] \mid \forall z\} \\
wlp_P(P, & x := y) & = & P \setminus \{[x\backslash z] \mid \forall z\} \quad \text{or } \Omega_P \text{ if } \exists z.[x\backslash z] \in P \wedge [y\backslash z] \notin P \\
wlp_P(P, & x := y.i) & = & P \setminus \{[x\backslash z] \mid \forall z\} \quad \text{or } \Omega_P \text{ if } \exists z.[x\backslash z] \in P \wedge [y\backslash z] \notin P \\
wlp_P(P, & x.i := y) & = & P \quad \text{or } \Omega_P \text{ if } \exists z.[x\backslash z] \in P \wedge [y\backslash z] \notin P \\
wlp_P(P, & C_1; C_2) & = & wlp_P(wlp_P(P, C_2), C_1) \\
wlp_P(P, & if\ B\ then\ C_1\ else\ C_2) & = & wlp_P(wlp_P(P, C_1), B) \cap wlp_P(P, C_2) \\
wlp_P(P, & if\ \neg B\ then\ C_1\ else\ C_2) & = & wlp_P(P, C_1) \cap wlp_P(wlp_P(P, C_2), B) \\
wlp_P(P, & if\ B\ then\ C_1) & = & wlp_P(wlp_P(P, C_1), B) \cap P \\
wlp_P(P, & if\ \neg B\ then\ C_1) & = & wlp_P(P, C_1) \cap wlp_P(P, B) \\
wlp_P(P, & while\ B\ do\ C_1) & = & gfp^{\subseteq}_{T_{op}}\ \lambda X.\ wlp_P(wlp_P(X, C_1), B) \cap P \\
wlp_P(P, & while\ \neg B\ do\ C_1) & = & gfp^{\subseteq}_{T_{op}}\ \lambda X.\ wlp_P(X, C_1) \cap wlp_P(P, B) \\
wp_P(P, & while\ B\ do\ C_1) & = & lfp^{\supseteq}_{P}\ \lambda X.\ wlp_P(wlp_P(X, C_1), B) \cap X \\
wp_P(P, & while\ \neg B\ do\ C_1) & = & lfp^{\supseteq}_{P}\ \lambda X.\ wlp_P(X, C_1) \cap wlp_P(X, B) \\
wlp_P(P, & \neg B) & = & P
\end{array}
$$

Where Ω_P is a partition error.

5.3 γ_P : Partitions → BI's formulas

Preliminary definitions

$$
\begin{array}{rcl}
\texttt{isloc}(x) & \equiv & \neg(x = \texttt{nil}) \wedge \neg(x = \texttt{true}) \wedge \neg(x = \texttt{false}) \wedge (\exists n.\ n = x + 1) \\
\texttt{isinheap}(x) & \equiv & \exists x_1, x_2.\ (x \hookrightarrow x_1, x_2) \\
\texttt{isdangling}(x) & \equiv & \texttt{isloc}(x) \wedge \neg\texttt{isinheap}(x) \\
\texttt{Nodangling2} & \equiv & \forall v, v'.\ (\texttt{isinheap}(v) \wedge (v' = v.1 \ \vee \ v' = v.2)) \Rightarrow \neg\ \texttt{isdangling}(v')
\end{array}
$$

$[x\backslash y]$ as a BI formula

$$
\begin{array}{rcl}
[x\backslash y] & \equiv & \neg\texttt{isloc}(x) \vee \neg\texttt{isloc}(y) \\
& & \vee \exists x_1, x_2, y_1, y_2. \\
& & \quad (x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}) \\
& & \quad *(y \hookrightarrow y_1, y_2 \wedge \texttt{Nodangling2}) \\
& & \quad *\texttt{Nodangling2}
\end{array}
$$

Notice that in case x or y reaches a dangling location, we do not have $[x\backslash y]$.

γ_P :Partitions → BI

$$\gamma_p(P) \quad = \quad \bigwedge_{[a\backslash b] \in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z\, \text{“}\in\text{”}\, P} \neg\ \texttt{isdangling}(z)$$

Since a partition is a finite set of $[a\backslash b]$ we can have this definition.

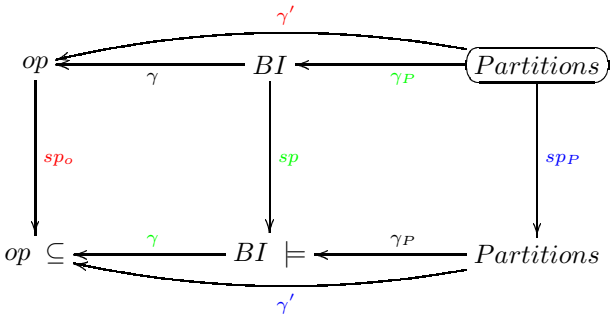
Notation : $\forall z\, \text{“}\in\text{”}\, P \equiv \exists w.[z\backslash w] \in P$.

To have some explanations of those definitions, see Appendix D.

5.4 α_P : BI's formulas $\rightarrow$ Partitions

We can not express $\alpha_P(F)$ by induction on the syntax of F. So we would have to check whether $F \models \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{ var of the program}} \neg\ \texttt{isdangling}(z)$, and if so $\alpha_P(F) = \{[a\backslash b] \mid F \models [a\backslash b]\}$.
Which means we will have to check whether $F \models [a\backslash b]$ for all pairs of variables.

5.5 Proof of the partition analysis



with $\gamma' = \gamma \circ \gamma_P$.

We prove the analysis by induction on the syntax of the commands.
So we have two ways for each command we can either prove that :

$$sp_o(\gamma'(P), C) \subseteq \gamma'(sp_P(P, C))$$

or prove that :

$$\gamma(sp(\gamma_P(P), C)) \subseteq \gamma'(sp_P(P, C))$$

since we already know that :

$$sp_o(\gamma(Q), C) \subseteq \gamma(sp(Q, C)) \ .$$

The partition analysis does not deal with errors, so in fact we only have :

$$sp_o(\gamma'(P), C) \setminus \{\Omega_o\} \subseteq \gamma'(sp_P(P, C))$$

we do not deal with errors in our proofs.

5.5.1 $x := \texttt{nil}$

$$\begin{aligned} sp_o(\gamma'(P), x := nil) &= \{s', h' \mid \exists s.h.\ s, h \models (\bigwedge_{[a\backslash b]\in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in\text{”} P} \neg\ \texttt{isdangling}(z)) \\ &\quad \wedge h' = h \wedge s' = s[x \mapsto \texttt{nil}]\} \\ \gamma'(sp_P(P, x := nil)) &= \{s', h' \mid s', h' \models \bigwedge_{\substack{[a\backslash b]\in P \\ a,b\neq x}} [a\backslash b] \wedge \bigwedge_{\forall z} [x\backslash z] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in\text{”} P} \neg\ \texttt{isdangling}(z)\} \end{aligned}$$

So we have to prove that if

$$s, h \models \bigwedge_{[a\backslash b]\in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in\text{”} P} \neg\ \texttt{isdangling}(z)$$

then

$$s[x \mapsto \texttt{nil}], h \models \bigwedge_{\substack{[a\backslash b]\in P \\ a,b\neq x}} [a\backslash b] \wedge \bigwedge_{\forall z} [x\backslash z] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in\text{”} P} \neg\ \texttt{isdangling}(z)$$

.

- We have $FV([a\backslash b]) = \{a, b\}$ so if $s, h \models [a\backslash b]$ for $a, b \neq x$ then $s[x \mapsto \texttt{nil}], h \models [a\backslash b]$.

- We have that $s[x \mapsto \texttt{nil}], h \models \neg\texttt{isloc}(x)$ so $s[x \mapsto \texttt{nil}], h \models \bigwedge_{\forall z}[x\backslash z]$.
- since $s, h \models \texttt{Nodangling2}$ we have $s[x \mapsto \texttt{nil}], h \models \texttt{Nodangling2}$
- since $s, h \models \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)$ we have $s[x \mapsto \texttt{nil}], h \models \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)$

5.5.2 $\texttt{new}(x)$

$$\begin{array}{rcl} sp_o(\gamma'(P), new(x)) & = & \{s', h' \mid \exists s.h.\ s, h \models (\bigwedge_{[a\backslash b]\in P}[a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)) \\ & & \wedge \exists l \in Loc. l \notin dom(h) \wedge h' = h[l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \wedge s' = s[x \mapsto l]\} \\ \gamma'(sp_P(P, new(x)) & = & \{s', h' \mid s', h' \models \bigwedge_{\substack{[a\backslash b]\in P \\ a,b\neq x}}[a\backslash b] \wedge \bigwedge_{\forall z,\ z\neq x}[x\backslash z] \\ & & \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)\} \end{array}$$

So we have to prove that if

$$s, h \models \bigwedge_{[a\backslash b]\in P}[a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)$$

then $\forall l \in Loc.\ l \notin dom(h)$

$$s[x \mapsto l], h[l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \models \bigwedge_{\substack{[a\backslash b]\in P \\ a,b\neq x}}[a\backslash b] \wedge \bigwedge_{\forall z,\ z\neq x}[x\backslash z] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)$$

.

- We have $FV([a\backslash b]) = \{a, b\}$ and $l \notin dom(h)$ so if $s, h \models [a\backslash b]$ for $a, b \neq x$ then $s[x \mapsto l], h[l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \models [a\backslash b]$.
- for $z \neq x$, if $s, h \models \neg\texttt{isloc}(z)$ then $s[x \mapsto l], h[l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \models \neg\texttt{isloc}(z)$ and $s[x \mapsto l], h[l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \models [x\backslash z]$
- for $z \neq x$, if $s, h \models isloc(z)$
  since $s, h \models \neg\texttt{isdangling}(z) \wedge \texttt{Nodangling2}$
  we have $s, h \models \exists z_1, z_2.\ z \hookrightarrow z_1, z_2 \wedge \texttt{Nodangling2}$
  and then $s[x \mapsto l], h \models \exists z_1, z_2.\ z \hookrightarrow z_1, z_2 \wedge \texttt{Nodangling2}$ (since there are no free variables in Nodangling2)
  we have $s[x \mapsto l], [l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \models \exists x_1, x_2.\ x \hookrightarrow x_1 x_2 \wedge \texttt{Nodangling2}$
  and since $s[x \mapsto l], \emptyset \models \texttt{Nodangling2}$ we have
  $s[x \mapsto l], h[l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \models [x\backslash z]$
- if $s, h \models \texttt{Nodangling2}$ then $s[x \mapsto l], h[l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \models \texttt{Nodangling2}$ since we can lose the property Nodangling2 only if the heap his reduced or if the cell add have some dangling2 (which is not the case of nil).
- if $s, h \models \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)$ then $s[x \mapsto l], h[l \mapsto \langle \texttt{nil}, \texttt{nil}\rangle] \models \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)$ since we did not reduce the heap and we just added x to the stack to a non dangling location

5.5.3 $x := y$

$$\begin{array}{rcl} sp_o(\gamma'(P), x := y) & = & \{s', h' \mid \exists s.h.\ s, h \models (\bigwedge_{[a\backslash b]\in P}[a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)) \\ & & \wedge h' = h \wedge s' = s[x \mapsto s(y)]\} \\ \gamma'(sp_P(P, x := y)) & = & \{s', h' \mid s', h' \models \bigwedge_{\substack{[a\backslash b]\in P \\ a,b\neq x}}[a\backslash b] \wedge \bigwedge_{[y\backslash z]\in P}[x\backslash z] \wedge \texttt{Nodangling2} \\ & & \wedge \bigwedge_{\forall z \text{“}\in'' P} \neg\,\texttt{isdangling}(z)\} \end{array}$$

So we have to prove that if

$$s,h \models \bigwedge_{[a\backslash b]\in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z\text{“}\in''P} \neg\, \texttt{isdangling}(z)$$

then

$$s[x \mapsto s(y)],h \models \bigwedge_{\substack{[a\backslash b]\in P\\ a,b\neq x}} [a\backslash b] \wedge \bigwedge_{[y\backslash z]\in P} [x\backslash z] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z\text{“}\in''P} \neg\, \texttt{isdangling}(z)$$

.

- We have $FV([a\backslash b]) = \{a,b\}$ so if $s,h \models [a\backslash b]$ for $a,b \neq x$ then $s[x \mapsto s(y)],h \models [a\backslash b]$.
- We have that $s,h \models \bigwedge_{[y\backslash z]\in P} [y\backslash z]$
  so we have $s[x \mapsto s(y)],h \models \exists x'.(\bigwedge_{[y\backslash z]\in P} [y\backslash z])[x'/x] \wedge x = y$ *(from BI's sp)*
  so we have $s[x \mapsto s(y)],h \models \bigwedge_{\substack{[y\backslash z]\in P\\ z\neq x}} [x\backslash z]$
- since $s,h \models \texttt{Nodangling2}$ we have $s[x \mapsto s(y)],h \models \texttt{Nodangling2}$
- since $s,h \models \neg\, \texttt{isdangling}(y)$
  we have $s[x \mapsto s(y)],h \models \neg\, \texttt{isdangling}(x)$
- since $s,h \models \bigwedge_{\substack{\forall z\text{“}\in''P\\ z\neq x}} \neg\, \texttt{isdangling}(z)$ we have $s[x \mapsto s(y)],h \models \bigwedge_{\substack{\forall z\text{“}\in''P\\ z\neq x}} \neg\, \texttt{isdangling}(z)$

5.5.4 $x := y.i$

$$\begin{aligned} sp_o(\gamma'(P), x := y.i) &= \{s',h' \mid \exists s.h.\ s,h \models (\bigwedge_{[a\backslash b]\in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z\text{“}\in''P} \neg\, \texttt{isdangling}(z)) \\ &\quad \wedge h' = h \wedge s' = s[x \mapsto \pi_i(h(s(y)))]\} \\ \gamma'(sp_P(P, x := y.i)) &= \{s',h' \mid s',h' \models \bigwedge_{\substack{[a\backslash b]\in P\\ a,b\neq x}} [a\backslash b] \wedge \bigwedge_{\substack{[y\backslash z]\in P\\ z\neq x}} [x\backslash z] \wedge \texttt{Nodangling2} \\ &\quad \wedge \bigwedge_{\forall z\text{“}\in''P} \neg\, \texttt{isdangling}(z)\} \end{aligned}$$

So we have to prove that if

$$s,h \models \bigwedge_{[a\backslash b]\in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z\text{“}\in''P} \neg\, \texttt{isdangling}(z)$$

then

$$s[x \mapsto \pi_i(h(s(y)))],h \models \bigwedge_{\substack{[a\backslash b]\in P\\ a,b\neq x}} [a\backslash b] \wedge \bigwedge_{[y\backslash z]\in P} [x\backslash z] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z\text{“}\in''P} \neg\, \texttt{isdangling}(z)$$

.

- We have $FV([a\backslash b]) = \{a,b\}$ so if $s,h \models [a\backslash b]$ for $a,b \neq x$ then $s[x \mapsto \pi_i(h(s(y)))],h \models [a\backslash b]$.
- We have that $s,h \models \bigwedge_{\substack{[y\backslash z]\in P\\ z\neq x}} [y\backslash z]$
  so we have $s[x \mapsto \pi_i(h(s(y)))],h \models \exists x'.(\bigwedge_{[y\backslash z]\in P} [y\backslash z])[x'/x] \wedge x = y.i$ *(from BI's sp)*
  so we have $s[x \mapsto \pi_i(h(s(y)))],h \models \bigwedge_{\substack{[y\backslash z]\in P\\ z\neq x}} [y\backslash z] \wedge x = y.i$
  we now want to prove that if $s[x \mapsto \pi_i(h(s(y)))],h \models [y\backslash z] \wedge x = y.i$ then $s[x \mapsto \pi_i(h(s(y)))],h \models [x\backslash z]$

if $s[x \mapsto \pi_i(h(s(y)))], h \models \neg isloc(z)$ then it's ok
if $\pi_i(h(s(y))) \notin Loc$ then it's ok
otherwise since $s[x \mapsto \pi_i(h(s(y)))], h \models [y \backslash z]$ we have
$s[x \mapsto \pi_i(h(s(y)))], h \models \exists y_1, y_2, z_1, z_2.\ (y \hookrightarrow y_1, y_2 \wedge \texttt{Nodangling2}) * (z \hookrightarrow z_1, z_2 \wedge \texttt{Nodangling2}) * \texttt{Nodangling2}$
and if $s', h' \models (y \hookrightarrow y_1, y_2 \wedge \texttt{Nodangling2})$ and $y_i \in Loc$ then from Nodangling2 we have that $y_i \in dom(h')$ and so $s', h' \models \exists y_{i1}, y_{i2}.\ (y_i \hookrightarrow y_{i1}, y_{i2} \wedge \texttt{Nodangling2})$
and then $s[x \mapsto \pi_i(h(s(y)))], h \models \exists y_{i1}, y_{i2}, z_1, z_2.\ (y_i \hookrightarrow y_{i1}, y_{i2} \wedge \texttt{Nodangling2}) * (z \hookrightarrow z_1, z_2 \wedge \texttt{Nodangling2}) * \texttt{Nodangling2}$
and so $s[x \mapsto \pi_i(h(s(y)))], h \models [x \backslash z]$

- since $s, h \models \texttt{Nodangling2}$ we have $s[x \mapsto \pi_i(h(s(y)))], h \models \texttt{Nodangling2}$
- since $s, h \models \texttt{Nodangling2}$ we have $s[x \mapsto \pi_i(h(s(y)))], h \models \neg \texttt{isdangling}(x)$
- since $s, h \models \bigwedge_{\substack{\forall z \text{“}\in''P \\ z \neq x}} \neg\ \texttt{isdangling}(z)$ we have $s[x \mapsto \pi_i(h(s(y)))], h \models \bigwedge_{\substack{\forall z \text{“}\in''P \\ z \neq x}} \neg\ \texttt{isdangling}(z)$

5.5.5 $x.1 := y$

$$
\begin{array}{rcl}
sp_o(\gamma'(P), x.1 := y) & = & \{s', h' \mid \exists s.h.\ s, h \models (\bigwedge_{[a\backslash b] \in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in''P} \neg\ \texttt{isdangling}(z)) \\
 & & \wedge h' = h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \wedge s' = s\} \\
\gamma'(sp_P(P, x.1 := y)) & = & \{s', h' \mid s', h' \models \bigwedge_{\substack{[a\backslash b] \in P \\ [a\backslash x] \in P \\ [b\backslash x] \in P}} [a\backslash b] \wedge \bigwedge_{\substack{[a\backslash b] \in P \\ [a\backslash y] \in P \\ [b\backslash y] \in P}} [a\backslash b] \wedge \bigwedge_{\substack{[a\backslash b] \in P \\ [a\backslash x] \in P \\ [a\backslash y] \in P}} [a\backslash b] \wedge \bigwedge_{\substack{[a\backslash b] \in P \\ [b\backslash x] \in P \\ [b\backslash y] \in P}} [a\backslash b] \\
 & & \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in''P} \neg\ \texttt{isdangling}(z)\}
\end{array}
$$

So we have to prove that if

$$s, h \models \bigwedge_{[a\backslash b] \in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in''P} \neg\ \texttt{isdangling}(z)$$

then

$$
s, h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \models \bigwedge_{\substack{[a\backslash b] \in P \\ [a\backslash x] \in P \\ [b\backslash x] \in P}} [a\backslash b] \wedge \bigwedge_{\substack{[a\backslash b] \in P \\ [a\backslash y] \in P \\ [b\backslash y] \in P}} [a\backslash b] \wedge \bigwedge_{\substack{[a\backslash b] \in P \\ [a\backslash x] \in P \\ [a\backslash y] \in P}} [a\backslash b] \wedge \bigwedge_{\substack{[a\backslash b] \in P \\ [b\backslash x] \in P \\ [b\backslash y] \in P}} [a\backslash b]
$$
$$
\wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z \text{“}\in''P} \neg\ \texttt{isdangling}(z)
$$

- Here we want to prove that if $s, h \models [a\backslash b] \wedge [a\backslash x] \wedge [b\backslash x]$
then $s, h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \models [a\backslash b]$

Lemma[1] Let us define

$$P(h_1, h_2) \equiv\ h = h_1.h_2\ \wedge\ s, h_1 \models \texttt{Nodangling2}\ \wedge\ l \in dom(h_1)$$

$|h_1|$ be the cardinal of $dom(h_1)$

If h_1 and h_1' are two heaps such that $\exists h_2.P(h_1, h_2)$ and $\exists h_2.P(h_1', h_2)$ hold and $|h_1|$ and $|h_1'|$ are minimals
then

$$h_1 = h_1'$$

Lemma[1]'s proof : Let h_1 and h'_1 be two heaps satisfying the hypothesis of the lemma. Suppose that $h_1 \neq h'_1$,

1- from the hypothesis's condition of smallnessity we have that $|h_1| = |h'_1|$

2- from our supposition of $h_1 \neq h'_1$, there exist an $l' \in dom(h_1)$ such that $l' \notin dom(h'_1)$.

3- Let split h_1 in $h_1 = h_{10}.h_{11}$ and split h'_1 in $h'_1 = h'_{10}.h'_{11}$ such that $h_{10} = h'_{10}$ and $\forall l'.\ l' \in dom(h_{11}) \Rightarrow l' \notin dom(h'_{11})$ and $\forall l'.\ l' \in dom(h'_{11}) \Rightarrow l' \notin dom(h_{11})$.

4- we have from *2-* that $|h_{10}| < h_1|$ and with also *1-* $|h'_{10}| < |h'_1|$

5- we have $l \in dom(h_{10})$ since both $l \in dom(h_1)$ and $l \in dom(h'_1)$

6- since h_1 was the smallest in term of size of the domain to satisfy the hypothesis and from *5-* we have that $s, h_{10} \not\models \texttt{Nodangling2}$

7- since $s, h_1 \models \texttt{Nodangling2}$, with *6-* we have $\exists l' \in dom(h_{10}),\ l'' \in dom(h_{11})$ and their exist a value a such that either $(l' \mapsto l'', a) \in h_{10}$ either $(l' \mapsto a, l'') \in h_{10}$.

8- from the definition of h_{11} we have $l'' \notin dom(h'_{11})$.

9- from *7-* and *3-* we have $l'' \notin dom(h_{10})$

10- since $h_{10} = h'_{10}$, $l'' \notin dom(h'_{10})$ and so $h'' \notin dom(h'_1)$

11- from *10-* and *7-* we have l'' is dangling2 in h'_1

and so there is a contradiction with the hypothesis. □.

Let's go back to what we want to prove :
if $s, h \models [a\backslash b] \wedge [a\backslash x] \wedge [b\backslash x]$
then $s, h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \models [a\backslash b]$

Let AB be $s, h \models [a\backslash b]$, AX be $s, h \models [a\backslash x]$ and BX be $s, h \models [b\backslash x]$ and C be $s, h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \models [a\backslash b]$.

We want $AB \wedge AX \wedge BX \Rightarrow C$.

If $s, h \models \neg\texttt{isloc}(a)$ or $s, h \models \neg\texttt{isloc}(b)$ we have directly C true. So from now on we will be in the case $s, h \not\models \neg\texttt{isloc}(a)$ and $s, h \not\models \neg\texttt{isloc}(b)$. We also have $s, h \not\models \neg\texttt{isloc}(x)$ otherwise we would not be able to run the command form s, h and the analysis is in case we can run the programm.

So we can unfold the definitions for AB into

$$\exists v_{a1}, v_{a2}, v_{b1}, v_{b2}, h_1, h_2, h_3.$$

- $h = h_1.h_2.h_3$
- $h_1(s(a)) = \langle v_{a1}, v_{a2}\rangle$
- $h_2(s(b)) = \langle v_{b1}, v_{b2}\rangle$
- $s[a1 \mapsto v_{a1}, a2 \mapsto v_{a2}, b1 \mapsto v_{b1}, [b2 \mapsto v_{b2}], h_1 \models \texttt{Nodangling2}$
- $s[a1 \mapsto v_{a1}, a2 \mapsto v_{a2}, b1 \mapsto v_{b1}, [b2 \mapsto v_{b2}], h_2 \models \texttt{Nodangling2}$
- $s[a1 \mapsto v_{a1}, a2 \mapsto v_{a2}, b1 \mapsto v_{b1}, [b2 \mapsto v_{b2}], h_3 \models \texttt{Nodangling2}$

and simplify it in :

$$\exists h_1, h_2, h_3.$$

- $h = h_1.h_2.h_3$
- $h_1(s(a)) = h(s(a))$
- $h_2(s(b)) = h(s(b))$

- $s, h_1 \models$ `Nodangling2`
- $s, h_2 \models$ `Nodangling2`
- $s, h_3 \models$ `Nodangling2`

The same way we have AX equivalent to

$$\exists h'_1, h'_2, h'_3.$$

- $h = h'_1.h'_2.h'_3$
- $h'_1(s(a)) = h(s(a))$
- $h'_3(s(x)) = h(s(x))$
- $s, h'_1 \models$ `Nodangling2`
- $s, h'_2 \models$ `Nodangling2`
- $s, h'_3 \models$ `Nodangling2`

BX equivalent to

$$\exists h''_1, h''_2, h''_3.$$

- $h = h''_1.h''_2.h''_3$
- $h''_2(s(b)) = h(s(b))$
- $h''_3(s(x)) = h(s(x))$
- $s, h''_1 \models$ `Nodangling2`
- $s, h''_2 \models$ `Nodangling2`
- $s, h''_3 \models$ `Nodangling2`

C equivalent to

$$\exists h^*_1, h^*_2, h^*_3.$$

- $h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] = h^*_1.h^*_2.h^*_3$
- $h^*_1(s(a)) = h(s(a))$
- $h^*_3(s(b)) = h(s(b))$
- $s, h^*_1 \models$ `Nodangling2`
- $s, h^*_2 \models$ `Nodangling2`
- $s, h^*_3 \models$ `Nodangling2`

Remark, here we can write for example that $h'_1(s(a)) = h(s(a))$ instead of $h'_1(s(a)) = h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle](s(a))$ since if $[a\backslash x]$ hold we have $s(a) \neq s(x)$.

Since we have AB, AX and BX we can chose some $h_1, h_2, h_3, h'_1, ...$ that satisfy the properties. Let's take them such that $|h_1|, |h_2|, |h'_1|, |h'_3|, |h''_2|, |h''_3|$ are the smallest as possible. From Lemma [1] we then have $h_1 = h'_1$, $h_2 = h''2$ and $h'_3 = h''_3$.

Now we have to find the h^*_1, h^*_2, h^*_3 that would make C true.

Let split the proof in 3 case :

Case 1 $s(y) \notin dom(h_1)$ and $s(y) \notin dom(h_2)$

Case 2 $s(y) \notin dom(h_1)$ and $s(y) \in dom(h_2)$

Case 3 $s(y) \in dom(h_1)$ and $s(y) \notin dom(h_2)$

Case 1 See Fig. 5.2 and Fig. 5.3 We can take :

$$h_1^* = h_1$$

$$h_2^* = h_2$$

$$h_3^* = h_3[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle]$$

To prove C the only not obvious (almost everything comes from AB) thing in this case is to prove that $s, h_3^* \models$ `Nodangling2` wich is

$$s, h_3[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \models \texttt{Nodangling2}$$

From AB we have that $s, h_3 \models$ `Nodangling2`.
So the $\neg$`Nodangling2` could only come from the $[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle]$ part of the heap. From AX we have that $s(x) \notin dom(h_1)$. From BX we have that $s(x) \notin dom(h_2'')$ which is also $s(x) \notin dom(h_2)$. So since $s(x) \in dom(h)$ (since we can run the command), we have $s(x) \in dom(h_3)$. And so the $\neg$`Nodangling2` could only come from the $s(y)$ in $[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle]$. But since we are in the case where $s(y) \notin dom(h_1)$ and $s(y) \notin dom(h_2)$ we have $s(y) \in dom(h_3)$ and so we have as expected

$$s, h_3^* \models \texttt{Nodangling2}$$

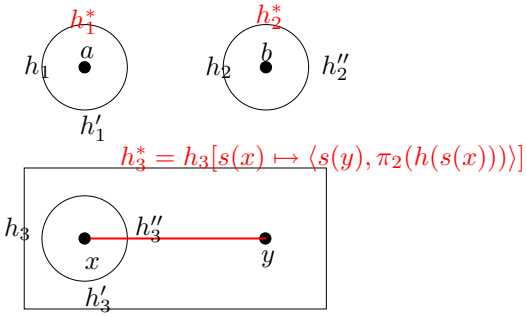


Figure 5.2: *Cas 1*

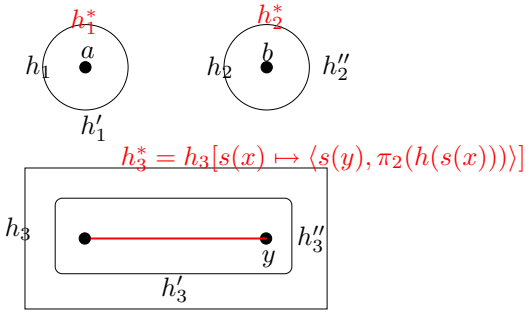


Figure 5.3: *Cas 1 bis*

Case 2 See Fig. 5.4 We can take :

$$h_1^* = h_1$$

$$h_2^* = h_3'.h_2[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle]$$

$$h_3^* = h_3 \backslash h_3'$$

To prove that with those h_1^*, h_2^*, h_3^* C hold, the first 3 points are obvious by construction.

$s, h_1^* \models$ `Nodangling2` from AB since $h_1^* = h_1$.

$s, h_2^* \models$ `Nodangling2` in the same as for the prove in the *Case 1* that $s, h_3^* \models$ `Nodangling2` : from AX we have $s, h_3' \models$ `Nodangling2`, from AB we have $s, h_2 \models$ `Nodangling2` so the $\neg$`Nodangling2` could only come from $[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle]$, and since $s(x) \in dom(h_3')$ the only dangling could be $s(y)$. But we are in the case where $s(y) \in dom(h_2)$ and so we have as expected

$$s, h_2^* \models \texttt{Nodangling2}$$

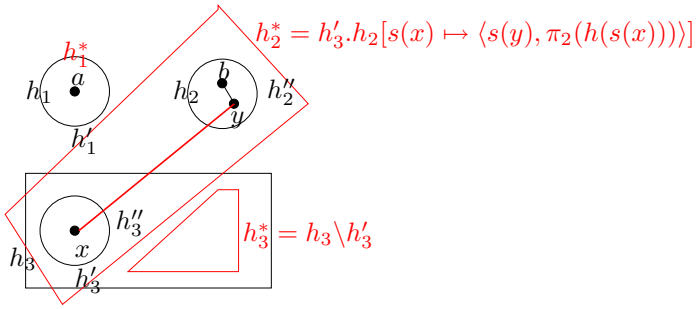


Figure 5.4: *Cas 2*

Now we still have to prove that $s, h_3^* \models \texttt{Nodangling2}$. We have $h_3 = h_3^*.h_3'$, from AB we have $s, h_3 \models \texttt{Nodangling2}$, from AX we have $s, h_3' \models \texttt{Nodangling2}$, so if we would have $s, h_3^* \models \neg\texttt{Nodangling2}$, there would be a location $l \in dom(h_3^*)$ such that $h(l) = \langle v_1, v_2\rangle$ with $v_1 \in dom(h_3')$ or $v_2 \in dom(h_3')$. But from AX we know that $s, h_2' \models \texttt{Nodangling2}$ and which is $s, h_2''.h_3^* \models \texttt{Nodangling2}$, but $h_3' = h_3''$ so if $v_i \in dom(h_3')$ we have $v_i \notin dom(h_2'')$ and so we would have $s, h_2''.h_3^* \not\models \texttt{Nodangling2}$.

Case 3 See Fig. 5.5 It's the same case as *Case 2* with a and b exchanged. We can take :

$$h_1^* = h_3'.h_1[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle]$$
$$h_2^* = h_2$$
$$h_3^* = h_3 \backslash h_3'$$

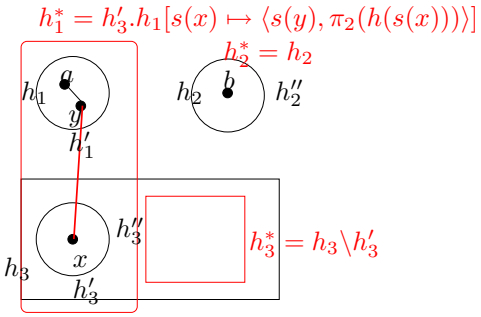


Figure 5.5: *Cas 3*

- Here we want to prove that if $s, h \models [a\backslash b] \wedge [a\backslash y] \wedge [b\backslash y]$
  same kind of proof as above
- Here we want to prove that if $s, h \models [a\backslash b] \wedge [a\backslash x] \wedge [a\backslash y]$
  same kind of proof as above
- Here we want to prove that if $s, h \models [a\backslash b] \wedge [b\backslash x] \wedge [b\backslash y]$
  same kind of proof as above.
- since $s, h \models \neg\, isdangling(y) \wedge \texttt{Nodangling2}$
  we have $s, h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \models \texttt{Nodangling2}$
- if $s, h \models \neg\, \texttt{isdangling}(z)$
  we have $s, h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \models \neg\, \texttt{isdangling}(z)$
  so we have $s, h[s(x) \mapsto \langle s(y), \pi_2(h(s(x)))\rangle] \models \bigwedge_{\forall z \text{“}\in''P} \neg\, \texttt{isdangling}(z)$

5.5.6 $C_1; C_2$

We prove that $sp_o(\gamma'(P), C_1; C_2) \subseteq \gamma'(sp_P(P, C_1; C_2))$ by induction on the size of the command.

$$\begin{array}{rcll} sp_o(\gamma'(P), C_1; C_2) & = & sp_o(sp_o(\gamma'(P), C_1), C_2) & definition \\ & \subseteq & sp_o(\gamma'(sp_P(P, C_1)), C_2) & induction\ hypothesis \\ & \subseteq & \gamma'(sp_P(sp_P(P, C_1)), C_2)) & induction\ hypothesis \\ & = & \gamma'(sp_P(P, C_1; C_2)) & definition \end{array}$$

5.5.7 $if\ \ B\ \ then\ \ C_1\ \ else\ \ C_2$

First notice some lemmas :

- [1] if $P_1 \subseteq P_2$ then $\gamma'(P_1) \supseteq \gamma'(P_2)$
- $sp_P(P, B) \subseteq P$

So we have that : $\gamma'(P) \subseteq \gamma'(sp_P(P, B))$
and so :

[3] $\gamma'(P) \cap \gamma(B = \texttt{true}) \subseteq \gamma'(sp_P(P, B))$.
We have :

- [4] if $\Delta_1 \subseteq \Delta_2$ then $sp_o(\Delta_1) \subseteq sp_o(\Delta_2)$

We will prove that :

$$sp_o(\gamma'(P), if\ \ B\ \ then\ \ C_1\ \ else\ \ C_2) \subseteq \gamma'(sp_P(P, if\ \ B\ \ then\ \ C_1\ \ else\ \ C_2))$$

by induction on the size of the command:

$$\begin{array}{rcll} \gamma'(sp_P(P, i\ B\ t\ C_1\ e\ C_2)) & = & \gamma'(sp_P(sp_P(P, B), C_1) \cap sp_P(P, C_2)) & \text{def. of } sp_P \\ & \supseteq & \gamma'(sp_P(sp_P(P, B), C_1)) \cup gamma'(sp_P(P, C_2)) & [1] \\ & \supseteq & sp_o(\gamma'(sp_P(P, B)), C_1) \cup sp_o(\gamma'(P), C_2) & \text{ind. hyp.} \\ & \supseteq & sp_o(\gamma'(P) \cap \gamma(B = \texttt{true}), C_1) \cup sp_o(\gamma'(P) \cap \gamma(B = \texttt{false}), C_2) & [3]\ [4] \\ & = & sp_o(\gamma'(P), if\ \ B\ \ then\ \ C_1\ \ else\ \ C_2) & \text{def. of } sp_o \end{array}$$

5.5.8 $if\ \ \neg\, B\ \ then\ \ C_1\ \ else\ \ C_2$

We can just notice that :

$$sp_P(P, if\ \ \neg\, B\ \ then\ \ C_1\ \ else\ \ C_2) = sp_P(P, if\ \ B\ \ then\ \ C_2\ \ else\ \ C_1)$$

and that

$$sp_o(\Delta, if\ \ \neg\, B\ \ then\ \ C_1\ \ else\ \ C_2) = sp_o(\Delta, if\ \ B\ \ then\ \ C_2\ \ else\ \ C_1)$$

and then then with the proof of :

$$sp_o(\gamma'(P), if\ \ B\ \ then\ \ C_2\ \ else\ \ C_1) \subseteq \gamma'(sp_P(P, if\ \ B\ \ then\ \ C_2\ \ else\ \ C_1))$$

we have

$$sp_o(\gamma'(P), if\ \ \neg\, B\ \ then\ \ C_1\ \ else\ \ C_2) \subseteq \gamma'(sp_P(P, if\ \ \neg\, B\ \ then\ \ C_1\ \ else\ \ C_2))$$

5.5.9 $if\ \ B\ \ then\ \ C_1$

This can be treated as a sub-case of $if\ \ B\ \ then\ \ C_1\ \ else\ \ C_2$ with `skip` as C_2.
`skip` is not in our language, but we can define it with $sp_o(\Delta, \texttt{skip}) = \Delta$ and extend sp_P with $sp_P(P, \texttt{skip}) = P$.
We then have $sp_o(\gamma'(P), \texttt{skip}) = \gamma'(sp_P(P, \texttt{skip}))$ and so $sp_o(\gamma'(P), \texttt{skip}) \subseteq \gamma'(sp_P(P, \texttt{skip}))$ as needed.

5.5.10 $if\ \neg B\ then\ C_1$

Same prove as the $if\ B\ then\ C_1$'s one.

5.5.11 $while\ B\ do\ C_1$

If we suppose that we have [1]:

$$lfp^{\subseteq}_{\gamma'(P)}\lambda X.\ \gamma'(Q)\{\gamma'(X) \to X\} \subseteq \gamma'(gfp^{\subseteq}_{P}\ \lambda X.\ Q)$$

. We can prove that

$$sp_o(\gamma'(P), while\ B\ do\ C_1) \subseteq \gamma'(sp_P(P, while\ B\ do\ C_1))$$

by induction on the size of the command.

$$\begin{array}{lcll}
\gamma'(sp_P(P, w\ B\ d\ C_1)) & = & \gamma'(gfp^{\subseteq}_{P}\lambda X.\ (sp_P(X, if\ B\ then\ C_1) \cap X)) & \text{def of } sp_P \\
& \supseteq & lfp^{\subseteq}_{\gamma'(P)}\lambda X.\ \gamma'(sp_P(X, if\ B\ then\ C_1))\{\gamma'(X) \to X\} \cup X & [1] \\
& \supseteq & lfp^{\subseteq}_{\gamma'(P)}\lambda X.\ sp_o(\gamma'(X), if\ B\ then\ C_1) \cup X & \text{ind hyp} \\
& \supseteq & (lfp\ \gamma'(P)^{\subseteq}\lambda X.\ sp_o(\gamma'(X), if\ B\ then\ C_1) \cup X) \cap \gamma(B = \texttt{false}) & \\
& = & sp_o(\gamma'(P), w\ B\ d\ C_1) & \text{def of } sp_o
\end{array}$$

So we have as expected

$$sp_o(\gamma'(P), while\ B\ do\ C_1) \subseteq \gamma'(sp_P(P, while\ B\ do\ C_1))$$

Lemma [1] If $lfp^{\subseteq}_{\gamma'(P)}\lambda X.\ \gamma'(Q)\{\gamma'(X) \to X\}$ and $gfp^{\subseteq}_{P}\ \lambda X.\ Q$ exists then

$$lfp^{\subseteq}_{\gamma'(P)}\lambda X.\ \gamma'(Q)\{\gamma'(X) \to X\} \subseteq \gamma'(gfp^{\subseteq}_{P}\ \lambda X.\ Q)$$

with $\{A \to B\}$ the syntactical substitution

Lemma's proof : let's define $U = gfp^{\subseteq}_{P}\ \lambda X.\ Q$ and $F = \lambda X.\ \gamma'(Q)\{\gamma'(X) \to X\}$

U is a fixpoint of $\lambda X.\ Q$,

$$\begin{array}{llcl}
\text{so} & U & \equiv & Q\{X \to U\} \\
\text{then} & \gamma'(U) & = & \gamma'(Q\{X \to U\}) \\
& & = & \gamma'(Q)\{\gamma'(X) \to \gamma'(U)\} \\
& & = & \gamma'(Q)\{\gamma'(X) \to X\}\{X \to \gamma'(U)\} \\
& & = & F(\gamma'(U))
\end{array}$$

and so $\gamma'(U)$ is a fixpoint of F and then $\gamma'(U) \supseteq lfp\ F$.□.

To have our proof finished, we should prove that for our use of lemma [1], the fixpoints exist. In our case $Q = sp_P(X, if\ B\ then\ C_1) \cap X$, so $\lambda X.\ Q$ is monotonic since sp_P is monotonic, and since the partition domain is a complete lattice (since it is sets over a finite domain), by Tarski's fixpoint theorem we have $gfp^{\subseteq}_{P}\ \lambda X.\ Q$ exists.
It remains to prove that the other fixpoint exists. If we add a top to the domain of the sets of memory, we have a complete lattice, and then we should just prove that the function $\lambda X.\ \gamma'(sp_P(X, if\ B\ then\ C_1))\{\gamma'(X) \to X\} \cup X$ is monotonic.

To finish, we should prove that

$$\gamma'(sp_P(X, if\ B\ then\ C_1))\{\gamma'(X) \to X\} \cup X$$

is equal to

$$\gamma'(sp_P(X, if\ B\ then\ C_1) \cap X)\{\gamma'(X) \to X\}$$

which is obvious since $\gamma'(A \cap B) = \gamma'(A) \cup \gamma'(B)$.

5.5.12 $while\ \neg\ B\ \ do\ \ C_1$

Proved the same way as for $while\ \ B\ \ do\ \ C_1$.

5.5.13 wlp's proofs

Not proved yet.

5.6 Remarks

Those definitions do not work for other version of BI where the locations are integers because of the way we have defined `isloc`. But the partition analysis is not defined for such a programming language.

Yang has defined some similar definitions like `allocated`(x), `noDangling`(x) and `noDanglingR` in his Ph.D. dissertation that are almost like `isinheap`(x), `isdangling`(x) and `Nodangling2` (he did not work with the integer version either).

Chapter 6

Conclusion

We are interested in BI-logic since it allows to work only in the part of the memory that is used by the program and the *Frame Axiom* let us do modular reasoning.

In chapter 4, we have given strongest post-conditions that could be used for a forward static analysis. We did not find forward analysis with BI in the literature.

In chapter 5, we have used BI as an interface for an other analysis. In the future, we would like to do it for different analyzes and in particular to abstract BI-logic into escape- and shape-analyzes.

Acknowledgment

Je remercie Radhia Cousot qui m'a accueillie dans son équipe ainsi que Patrick Cousot pour leurs conseils.

Je remercie mes camarades de bureau Charles, Damien et Francesco pour leurs conseils. Je remercie particulièrement Charles et Bruno pour leurs corrections lors de la rédaction de ce rapport.

Appendix A

Frame axiom's explanation

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \quad \mathit{ModifiesOnly}(C) \cap FV(R) = \emptyset$$

$\mathit{ModifiesOnly}(C)$: set of variables appearing to the left of := in C and not dereferenced.

Formal proof Yang and O'Hearn gave a formal proof of the Frame Axiom in [8].

Here I am just trying to give an "intuition" of why this restrictions $\mathit{ModifiesOnly}(C) \cap FV(R) = \emptyset$.

Why this restriction is needed ?

$$\begin{array}{rl} \{(y \hookrightarrow 1,2)\}x := y\{(y \hookrightarrow 1,2)\} & \text{true} \\ \{(y \hookrightarrow 1,2) * (x \hookrightarrow 3,4)\}x := y\{(y \hookrightarrow 1,2) * (x \hookrightarrow 3,4)\} & \text{false} \end{array}$$

Why this restriction is enough ?

- no need of a x from $x.i$
  because then P is necessary of the form of $(x \mapsto x_1, x_2) * P'$ for C to executable
  and then if x is in R and R intersects with P and $P * R$ is never satisfied and it's OK.

- no need of a y from $x := y$ because
  if y is not a pointer it will be only modified by a $y := ...$ and then it will appear in the restriction
  if y is a pointer, it can not be modified by a $x := ...$ and then to be modified by a $x.i := ...$
  it implies that we already had the P of the shape of $(y \mapsto x_1, x_2) * P'$ and we can conclude like the previous case

- Why can't we modify a variable not mentioned in C but in P ? Because we do not have $E.i.j$ or $E.i = E'.j$

Appendix B

sp's proofs

B.1 $x := E$

$$
\begin{array}{rcllll}
sp_o(\gamma(P), x := E) & = & \{s', h' \mid \exists s, h. & s, h \models & P \wedge\ h' = h \wedge\ s' = s[x \mapsto [\![E]\!]s]\} \\
\gamma(sp(P, x := E)) & = & \{s', h' \mid & s', h' \models & \exists x'.\ P[x/x] \wedge\ x = E[x'/x]\} \\
 & = & \{s', h' \mid & \exists v & ((\exists s_1, h_1 \\
 & & & & \quad \wedge\ s_1, h_1 \models P \\
 & & & & \quad \wedge\ h1 = h' \\
 & & & & \quad \wedge\ s_1 = s'[x' \mapsto v][x \mapsto v]) \\
 & & & & \wedge\ (\exists s_2, h_2 \\
 & & & & \quad \wedge\ h2 = h' \\
 & & & & \quad \wedge\ s'(x) = [\![E]\!]s_2 \\
 & & & & \quad \wedge\ s_2 = s'[x' \mapsto v][x \mapsto v]))))\}
\end{array}
$$

We can prove the inclusion by taking $v = s(x)$ if $x \in dom(s)$ and any value otherwise, $s_1, h_1 = s_2, h_2 = s[x' \mapsto v][x \mapsto v], h$.

We could also prove the inclusion in the other way by taking $s, h = s'[x \mapsto v], h'$.

B.2 $x := E.i$

$$
\begin{array}{rcllll}
sp_o(\gamma(P), x := E.i) & = & \{s', h' \mid \exists s, h. & s, h \models & P \wedge\ h' = h \wedge\ [\![E]\!]s \in Loc\ \wedge\ (\exists v.\ v = \pi_i(h([\![E]\!]s)) \wedge\ s' = s[x \mapsto v])\} \\
\gamma(sp(P, x := E.i)) & = & \{s', h' \mid & s', h' \models & \exists x'.\ P[x'/x] \wedge\ x = (E[x'/x]).i\} \\
 & = & \{s', h' \mid & \exists v_2 & ((\exists s_1, h_1 \\
 & & & & \quad \wedge\ s_1, h_1 \models P \\
 & & & & \quad \wedge\ h1 = h' \\
 & & & & \quad \wedge\ s_1 = s'[x' \mapsto v_2][x \mapsto v_2]) \\
 & & & & \wedge\ (\exists s_2, h_2 \\
 & & & & \quad \wedge\ h2 = h' \\
 & & & & \quad \wedge\ s'(x) = \pi_i(h_2([\![E]\!]s_2)) \\
 & & & & \quad \wedge\ s_2 = s'[x' \mapsto v_2][x \mapsto v_2]))))\}
\end{array}
$$

We can prove the inclusion by taking $v_2 = s(x)$ if $x \in dom(s)$ and any value otherwise, $s_1, h_1 = s_2, h_2 = s[x' \mapsto v_2][x \mapsto v_2], h$.

We could also prove the inclusion in the other way by taking $s, h = s'[x \mapsto v_2], h'$.

B.3 $E_1.i := E_2$

Not typed yet.

B.4 $x := \texttt{cons}(E_1, E_2)$

Not typed yet.

B.5 $\texttt{dispose}(E)$

Not typed yet.

B.6 $C_1; C_2$

We prove that $sp_o(\gamma(P), C_1; C_2) \subseteq \gamma(sp(P, C_1; C_2))$ by induction on the size of the command.

$$\begin{array}{lcll} sp_o(\gamma(P), C_1; C_2) & = & sp_o(sp_o(\gamma(P), C_1), C_2) & definition \\ & \subseteq & sp_o(\gamma(sp(P, C_1)), C_2) & induction\ hypothesis \\ & \subseteq & \gamma(sp((sp(P, C_1)), C_2)) & induction\ hypothesis \\ & = & \gamma(sp(P, C_1; C_2)) & definition \end{array}$$

B.7 $if\ \ E\ \ then\ \ C_1\ \ else\ \ C_2$

$C = if\ \ E\ \ then\ \ C_1\ \ else\ \ C_2$

$$\begin{array}{lcccll} sp_o(\gamma(P), C) & = & \{s', h' \mid \exists s, h. & s, h \models P & \wedge ((\llbracket E \rrbracket s = True\ \wedge\ s', h' \in sp_o(\{s, h\}, C_1)) \\ & & & & \vee (\llbracket E \rrbracket s = False\ \wedge\ s', h' \in sp_o(\{s, h\}, C_2)))\} \\ & = & \{s', h' \mid \exists s, h. & ((s, h \models P & \wedge E = \texttt{true}\ \wedge\ s', h' \in sp_o(\{s, h\}, C_1)) \\ & & & \vee (s, h \models P & \wedge E = \texttt{false}\ \wedge\ s', h' \in sp_o(\{s, h\}, C_2)))\} \\ & = & sp_o(\gamma(P \wedge E = \texttt{true}), C_1) & \cup & sp_o(\gamma(P \wedge E = \texttt{false}), C_2) \\ \gamma(sp(P, C)) & = & \{s', h' \mid & s', h' \models & (sp(P \wedge E = \texttt{true}, C_1) \\ & & & & \vee sp(P \wedge E = \texttt{false}, C_2))\} \\ & = & \{s', h' \mid & s', h' \models & sp(P \wedge E = \texttt{true}, C_1)\} \\ & & \cup \{s', h' \mid & s', h' \models & sp(P \wedge E = \texttt{false}, C_2)\} \\ & = & \gamma(sp(P \wedge E = \texttt{true}, C_1)) & \cup & \gamma(sp(P \wedge E = \texttt{false}, C_2)) \end{array}$$

We prove by induction in the size of the command.

B.8 $if\ \ E\ \ then\ \ C_1$

$C = if\ \ E\ \ then\ \ C_1$

$$\begin{array}{lcccll} sp_o(\gamma(P), C) & = & \{s', h' \mid \exists s, h. & s, h \models P & \wedge ((\llbracket E \rrbracket s = True\ \wedge\ s', h' \in sp_o(\{s, h\}, C_1)) \\ & & & & \vee (\llbracket E \rrbracket s = False\ \wedge\ s', h' = s, h))\} \\ & = & \{s', h' \mid \exists s, h. & ((s, h \models P & \wedge E = \texttt{true}\ \wedge\ s', h' \in sp_o(\{s, h\}, C_1)) \\ & & & \vee (s, h \models P & \wedge E = \texttt{false}\ \wedge\ s', h' = s, h))\} \\ & = & sp_o(\gamma(P \wedge E = \texttt{true}), C_1) & \cup & \gamma(P \wedge E = \texttt{false}) \\ \gamma(sp(P, C)) & = & \{s', h' \mid \exists s, h. & s, h \models & (sp(P \wedge E = \texttt{true}, C_1) \\ & & & & \vee (P \wedge E = \texttt{false}))\} \\ & = & \{s', h' \mid & s', h' \models & sp(P \wedge E = \texttt{true}, C_1)\} \\ & & \cup \{s', h' \mid & s', h' \models & P \wedge E = \texttt{false}\} \\ & = & \gamma(sp(P \wedge E = \texttt{true}, C_1)) & \cup & \gamma(P \wedge E = \texttt{false}) \end{array}$$

We prove by induction in the size of the command.

B.9 *while E do C*$_1$

Lemma If $lfp \overset{\models}{}_P \lambda X.\ Q$ and $lfp \overset{\subseteq}{}_{\gamma(P)} \lambda X.\ \gamma(Q)\{\gamma(X) \to X\}$ exists then $\gamma(lfp \overset{\models}{}_P \lambda X.\ Q) \supseteq lfp \overset{\subseteq}{}_{\gamma(P)} \lambda X.\ \gamma(Q)\{\gamma(X) \to X\}$

with $\{A \to B\}$ the syntactical substitution

Lemma's proof : let's define $U = lfp \overset{\models}{}_P \lambda X.\ Q$ and $F = \lambda X.\ \gamma(Q)\{\gamma(X) \to X\}$

U is a fixpoint of $\lambda X.\ Q$,

$$\begin{array}{lrcl} \text{so} & U & \equiv & Q\{X \to U\} \\ \text{then} & \gamma(U) & = & \gamma(Q\{X \to U\}) \\ & & = & \gamma(Q)\{\gamma(X) \to \gamma(U)\} \\ & & = & \gamma(Q)\{\gamma(X) \to X\}\{X \to \gamma(U)\} \\ & & = & F(\gamma(U)) \end{array}$$

and so $\gamma(U)$ is a fixpoint of F and then $\gamma(U) \supseteq lfp\ F$.□

$$\begin{array}{lcll} sp_o(\gamma(P), w\ E\ d\ C_1) & = & (lfp \overset{\subseteq}{}_{\gamma(P)} \lambda X. & \{s', h' \mid \exists s, h.\quad s, h \in X \wedge \quad ((\llbracket E \rrbracket s = True\ \wedge s', h' \in sp_o(\{s, h\}, C_1)) \\ & & & \qquad \vee (s', h' = s, h))\}) \\ & & \cap \{s', h' \mid \llbracket E \rrbracket s' = \texttt{false}\} & \\ & = & (lfp \overset{\subseteq}{}_{\gamma(P)} \lambda X. & \{s', h' \mid \exists s, h.\quad s, h \in X \wedge \quad \llbracket E \rrbracket s = True\ \wedge s', h' \in sp_o(\{s, h\}, C_1)\} \\ & & \cup X) & \\ & & \cap\ \gamma(E = \texttt{false}) & \\ & = & (lfp \overset{\subseteq}{}_{\gamma(P)} \lambda X. & sp_o(X \cap \gamma(E = \texttt{false}), C_1) \cup X) \\ & & \cap\ \gamma(E = \texttt{false}) & \\ \gamma(sp(P, w\ E\ d\ C_1)) & = & \gamma(lfp \overset{\models}{}_P \lambda X. & (sp(X \wedge E = \texttt{true}, C_1) \vee X) \\ & & \quad \wedge (E = \texttt{false})) & \\ & = & \gamma(lfp \overset{\models}{}_P \lambda X. & (sp(X \wedge E = \texttt{true}, C_1) \vee X)) \\ & & \cap\ \gamma(E = \texttt{false}) & \end{array}$$

So by induction on the size of the command and by the lemma we have that if the lfp s exist, $sp_o(\gamma(P), while\ \ E\ \ do\ \ C_1) \subseteq \gamma(sp(P, while\ \ E\ \ do\ \ C_1))$.

And if the lfp does not exist, $\gamma(sp(P, while\ \ E\ \ do\ \ C_1)) = \gamma(E = \texttt{false})$ in which $sp_o(\gamma(P), while\ \ E\ \ do\ \ C_1)$ is included since the $\cap$ and so we have $sp_o(\gamma(P), while\ \ E\ \ do\ \ C_1) \subseteq \gamma(sp(P, while\ \ E\ \ do\ \ C_1))$.

Appendix C

sp^2's proofs

C.1 $x := E$

$$
\begin{array}{lcll}
sp^2_o(\gamma^2(P), x := E) & = & \{m' \mid \exists m.\ m \models^2 P \wedge & ((m' = \Omega_o \wedge & (\ m = \Omega_o \\
 & & & & \vee (m = s, h \wedge\ m \models^2 \neg(E = E)))) \\
 & & & \vee (m' = s', h' & \wedge\ m = s, h \\
 & & & & \wedge\ h' = h \wedge\ s' = s[x \mapsto [\![E]\!]s]))\} \\
 & = & \{m' \mid \exists m.\ m \models^2 P \wedge & (m' = \Omega_o \wedge & (\ m = \Omega_o \\
 & & & & \vee (m = s, h \wedge\ m \models^2 \neg(E = E))))\} \\
 & & \cup \{m' \mid \exists m.\ m \models^2 P \wedge & (m' = s', h' \wedge & m = s, h \\
 & & & & \wedge\ h' = h \wedge\ (\exists v.\ v = [\![E]\!]s \wedge\ s' = s[x \mapsto v]))\} \\
\gamma^2(sp^2(P, x := E)) & = & \{m' \mid m' \models^2 & ((\texttt{Err} \wedge & (P \\
 & & & & \vee (\exists^2 m.\ (P \wedge\ \neg(E = E))))) \\
 & & & \vee (\exists x'.\ P[x'/x] & \wedge\ x = (E[x'/x])))\} \\
 & = & \{m' \mid m' \models^2 & (\texttt{Err} \wedge & (P \\
 & & & & \vee (\exists^2 m.\ (P \wedge\ \neg(E = E)))))\} \\
 & & \cup \{m' \mid m' \models^2 & \exists x'.\ P[x'/x] & \wedge\ x = (E[x'/x]))\} \\
 & = & \{m' \mid \exists m.\ m \models^2 P \wedge & (m' = \Omega_o \wedge & (\ m = \Omega_o \\
 & & & & \vee (m = s, h \wedge\ m \models^2 \neg(E = E))))\} \\
 & & \cup \{m' \mid m' = s', h' \wedge & \exists v & ((\exists m_1.\ m_1 = s_1, h_1 \\
 & & & & \quad \wedge\ m_1 \models^2 P \\
 & & & & \quad \wedge\ h1 = h' \\
 & & & & \quad \wedge\ s_1 = s'[x' \mapsto v][x \mapsto v]) \\
 & & & & \wedge\ (\exists m_2.\ m_2 = s_2, h_2 \\
 & & & & \quad \wedge\ h2 = h' \\
 & & & & \quad \wedge\ s'(x) = [\![E]\!]s_2 \\
 & & & & \quad \wedge\ s_2 = s'[x' \mapsto v][x \mapsto v]))))\}
\end{array}
$$

Remarque that we write $\neg(E = E)$ as a condition for E not to be defined.

We have the red parts equals (error cases).

So we need to prove that the blue part of $sp^2_o(\gamma^2(P), x := E)$ is included in the blue part of $\gamma^2(sp^2(P, x := E))$. We can prove it by taking $v = s(x)$ if $x \in dom(s)$ and any value otherwise, $m1 = m2 = s[x' \mapsto v][x \mapsto v], h$.

We could also prove the inclusion in the other way by taking $m = s'[x \mapsto v], h'$.

C.2 $x := E.i$

$$
\begin{array}{lcll}
sp^2_o(\gamma^2(P), x := E.i) & = & \{m' \mid \exists m.\ m \models^2 P \wedge & ((m' = \Omega_o \wedge \\
 & & & \quad (\ m = \Omega_o \\
 & & & \quad \vee (m = s, h \\
 & & & \qquad \wedge\ m \models^2 \neg(\exists x_1, x_2.\ E \hookrightarrow x_1, x_2)))) \\
 & & & \vee (m' = s', h' \quad \wedge\ m = s, h \\
 & & & \qquad \wedge\ h' = h \\
 & & & \qquad \wedge\ [\![E]\!]s \in Loc \\
 & & & \qquad \wedge\ (\exists v.\ v = \pi_i(h([\![E]\!]s)) \wedge\ s' = s[x \mapsto v]))\} \\
 & = & \{m' \mid \exists m.\ m \models^2 P \wedge & (m' = \Omega_o \wedge \quad (\ m = \Omega_o \\
 & & & \quad \vee (m = s, h \\
 & & & \qquad \wedge\ m \models^2 \neg(\exists x_1, x_2.\ E \hookrightarrow x_1, x_2)))\} \\
 & & \cup \{m' \mid \exists m.\ m \models^2 P \wedge & (m' = s', h' \wedge \quad m = s, h \\
 & & & \qquad \wedge\ h' = h \\
 & & & \qquad \wedge\ [\![E]\!]s \in Loc \\
 & & & \qquad \wedge\ (\exists v.\ v = \pi_i(h([\![E]\!]s)) \wedge\ s' = s[x \mapsto v]))\} \\
\gamma^2(sp^2(P, x := E)) & = & \{m' \mid m' \models^2 & ((\texttt{Err} \wedge \quad (P \\
 & & & \quad \vee (\exists^2 m.\ (P \wedge\ \neg(\exists x_1, x_2.\ E \hookrightarrow x_1, x_2))))) \\
 & & & \vee (\exists x'.\ P[x'/x] \quad \wedge\ x = (E[x'/x]).i))\} \\
 & = & \{m' \mid m' \models^2 & (\texttt{Err} \wedge \quad (P \\
 & & & \quad \vee (\exists^2 m.\ (P \wedge\ \neg(\exists x_1, x_2.\ E \hookrightarrow x_1, x_2))))))\} \\
 & & \cup \{m' \mid m' \models^2 & \exists x'.\ P[x'/x] \quad \wedge\ x = (E[x'/x]).i)\} \\
 & = & \{m' \mid \exists m.\ m \models^2 P \wedge & (m' = \Omega_o \wedge \quad (\ m = \Omega_o \\
 & & & \quad \vee (m = s, h \\
 & & & \qquad \wedge\ m \models^2 \neg(\exists x_1, x_2.\ E \hookrightarrow x_1, x_2)))\} \\
 & & \cup \{m' \mid m' = s', h' \wedge & \exists v_2 \quad ((\exists m_1.\ m_1 = s_1, h_1 \\
 & & & \qquad \wedge\ m_1 \models^2 P \\
 & & & \qquad \wedge\ h1 = h' \\
 & & & \qquad \wedge\ s_1 = s'[x' \mapsto v_2][x \mapsto v_2]) \\
 & & & \quad \wedge\ (\exists m_2.\ m_2 = s_2, h_2 \\
 & & & \qquad \wedge\ h2 = h' \\
 & & & \qquad \wedge\ s'(x) = \pi_i(h_2([\![E]\!]s_2)) \\
 & & & \qquad \wedge\ s_2 = s'[x' \mapsto v_2][x \mapsto v_2]))\}
\end{array}
$$

We have the red parts equals (error cases).

So we need to prove that the blue part of $sp^2_o(\gamma^2(P), x := E)$ is included in the blue part of $\gamma^2(sp^2(P, x := E))$. We can prove it by taking $v_2 = s(x)$ if $x \in dom(s)$ and any value otherwise, $m1 = m2 = s[x' \mapsto v_2][x \mapsto v_2], h$.

We could also prove the inclusion in the other way by taking $m = s'[x \mapsto v_2], h'$.

C.3 $E_1.i := E_2$

Not typed yet.

C.4 $x := \texttt{cons}(E_1, E_2)$

Not typed yet.

C.5 $\texttt{dispose}(E)$

Not typed yet.

C.6 $C_1; C_2$

We prove that $sp_o^2(\gamma^2(P), C_1; C_2) \subseteq \gamma^2(sp^2(P, C_1; C_2))$ by induction on the size of the command.

$$\begin{array}{lcll} sp_o^2(\gamma^2(P), C_1; C_2) & = & sp_o^2(sp_o^2(\gamma^2(P), C_1), C_2) & definition \\ & \subseteq & sp_o^2(\gamma^2(sp^2(P, C_1)), C_2) & induction\ hypothesis \\ & \subseteq & \gamma^2(sp^2((sp^2(P, C_1)), C_2)) & induction\ hypothesis \\ & = & \gamma^2(sp^2(P, C_1; C_2)) & definition \end{array}$$

C.7 $if\ E\ then\ C_1\ else\ C_2$

$C = if\ E\ then\ C_1\ else\ C_2$

$$
\begin{array}{lcll}
sp^2_o(\gamma^2(P), C) & = & \{m' \mid \exists m. & m \models^2 P \\
& & & \wedge((m' = \Omega_o \wedge \\
& & & \quad (\ m = \Omega_o \\
& & & \quad \vee (m = s, h \\
& & & \qquad \wedge\, m \models^2 \neg(E = E)))) \\
& & & \vee (m = s, h \\
& & & \quad \wedge [\![E]\!]s = True \\
& & & \qquad \wedge m' \in sp^2_o(m, C_1)) \\
& & & \vee (m = s, h \\
& & & \quad \wedge [\![E]\!]s = False \\
& & & \qquad \wedge m' \in sp^2_o(m, C_2)))\} \\
& = & \{m' \mid \exists m. & ((m \models^2 P \\
& & & \wedge (m' = \Omega_o \wedge \\
& & & \quad (\ m = \Omega_o \\
& & & \quad \vee (m = s, h \\
& & & \qquad \wedge\, m \models^2 \neg(E = E))))) \\
& & & \vee (m \models^2 P \\
& & & \quad \wedge\ m = s, h \\
& & & \qquad \wedge m' \in sp^2_o(m, C_1)) \\
& & & \wedge [\![E]\!]s = True \\
& & & \vee (m \models^2 P \\
& & & \quad \wedge\ m = s, h \\
& & & \qquad \wedge m' \in sp^2_o(m, C_2)))\} \\
& = & \{m' \mid \exists m. & ((m \models^2 P \\
& & & \wedge (m' = \Omega_o \wedge \\
& & & \quad (\ m = \Omega_o \\
& & & \quad \vee (m = s, h \\
& & & \qquad \wedge\, m \models^2 \neg(E = E))))) \\
& & & \vee (m \models^2 P \wedge E = \texttt{true} \\
& & & \quad \wedge\ m' \in sp^2_o(m, C_1)) \\
& & & \vee (m \models^2 P \wedge E = \texttt{false} \\
& & & \quad \wedge\ m' \in sp^2_o(m, C_2)))\} \\
\gamma^2(sp^2(P, C)) & = & \{m' \mid & m' \models^2 \\
& & & ((\texttt{Err} \wedge \\
& & & \quad (P \vee \\
& & & \quad (\exists^2 m.\ (P \wedge\ \neg(E = E))))) \\
& & & \vee sp^2(P \wedge E = \texttt{true}, C_1) \\
& & & \vee sp^2(P \wedge E = \texttt{false}, C_2))\} \\
& = & \{m' \mid & (\exists m.\ m \models^2 P \\
& & & \wedge (m' = \Omega_o \wedge \\
& & & \quad (\ m = \Omega_o \\
& & & \quad \vee (m = s, h \\
& & & \qquad \wedge\, m \models^2 \neg(E = E))))) \\
& & & \vee m' \models^2 \\
& & & sp^2(P \wedge E = \texttt{true}, C_1) \\
& & & \vee m' \models^2 \\
& & & sp^2(P \wedge E = \texttt{false}, C_2))\} \\
& = & \{m' \mid & (\exists m.\ m \models^2 P \\
& & & \wedge (m' = \Omega_o \wedge \\
& & & \quad (\ m = \Omega_o \\
& & & \quad \vee (m = s, h \\
& & & \qquad \wedge\, m \models^2 \neg(E = E)))))\} \\
& & \cup & \gamma^{2(} sp^2(P \wedge E = \texttt{true}, C_1)) \\
& & \cup & \gamma^{2(} sp^2(P \wedge E = \texttt{false}, C_2))
\end{array}
$$

We prove by "color" cases and for blue and green by induction in the size of the command.
If m' is in $sp^2_o(\gamma^2(P), if\ E\ then\ C_1\ else\ C_2)$, then there is a m such that :
- we are in the red case, then m' is in $\gamma^2(sp^2(P, if\ E\ then\ C_1\ else\ C_2))$ by his red set.
- we are in the blue case then $m' \in sp^2_o(\gamma^2(P \wedge E = \texttt{true}), C_1)$, then by induction hypothesis m' is in the blue set of $\gamma^2(sp^2(P, if\ E\ then\ C_1\ else\ C_2))$
- we are in the green case then $m' \in sp^2_o(\gamma^2(P \wedge E = \texttt{false}), C_1)$, then by induction hypothesis m' is in the green set of $\gamma^2(sp^2(P, if\ E\ then\ C_1\ else\ C_2))$

We can conclude that $sp^2_o(\gamma^2(P), if\ E\ then\ C_1\ else\ C_2) \subseteq \gamma^2(sp^2(P, if\ E\ then\ C_1\ else\ C_2))$

C.8 $if\ \ E\ \ then\ \ C_1$

$C = if\ \ E\ \ then\ \ C_1$

$$
\begin{array}{lcll}
sp^2_o(\gamma^2(P), C) & = & \{m' \mid \exists m. \quad m \models^2 P & \wedge((m' = \Omega_o \wedge & (\ m = \Omega_o \\
 & & & & \vee(m = s, h \\
 & & & & \quad \wedge\, m \models^2 \neg(E = E)))) \\
 & & & \vee(m = s, h & \wedge [\![E]\!]s = True \\
 & & & & \quad \wedge m' \in sp^2_o(m, C_1)) \\
 & & & \vee(m = s, h & \wedge [\![E]\!]s = False \\
 & & & & \quad \wedge m' = m))\} \\
 & = & \{m' \mid \exists m. \quad ((m \models^2 P & \wedge(m' = \Omega_o \wedge & (\ m = \Omega_o \\
 & & & & \vee(m = s, h \\
 & & & & \quad \wedge\, m \models^2 \neg(E = E))))) \\
 & & \vee(m \models^2 P & \wedge\, m = s, h & \wedge [\![E]\!]s = True \\
 & & & \quad \wedge m' \in sp^2_o(m, C_1)) & \\
 & & \vee(m \models^2 P & \wedge\, m = s, h & \wedge [\![E]\!]s = False \\
 & & & \quad \wedge m' = m))\} & \\
 & = & \{m' \mid \exists m. \quad ((m \models^2 P & \wedge(m' = \Omega_o \wedge & (\ m = \Omega_o \\
 & & & & \vee(m = s, h \\
 & & & & \quad \wedge\, m \models^2 \neg(E = E))))) \\
 & & \vee(m \models^2 P \wedge E = \texttt{true} & \wedge\, m' \in sp^2_o(m, C_1)) & \\
 & & \vee(m \models^2 P \wedge E = \texttt{false} & \wedge\, m' = m))\} & \\
\gamma^2(sp^2(P, C)) & = & \{m' \mid \quad m' \models^2 & ((\texttt{Err} \wedge & (P \vee \\
 & & & & (\exists^2 m.\ (P \wedge\ \neg(E = E))))) \\
 & & & \vee sp^2(P \wedge E = \texttt{true}, C_1) & \\
 & & & \vee P \wedge E = \texttt{false}))\} & \\
 & = & \{m' \mid \quad (\exists m.\ m \models^2 P & \wedge(m' = \Omega_o \wedge & (\ m = \Omega_o \\
 & & & & \vee(m = s, h \\
 & & & & \quad \wedge\, m \models^2 \neg(E = E))))) \\
 & & \vee m' \models^2 & sp^2(P \wedge E = \texttt{true}, C_1) & \\
 & & \vee m' \models^2 & (P \wedge E = \texttt{false}))\} & \\
 & = & \{m' \mid \quad (\exists m.\ m \models^2 P & \wedge(m' = \Omega_o \wedge & (\ m = \Omega_o \\
 & & & & \vee(m = s, h \\
 & & & & \quad \wedge\, m \models^2 \neg(E = E)))))\} \\
 & & \cup & \gamma^{2(} sp^2(P \wedge E = \texttt{true}, C_1)) & \\
 & & \cup & \gamma^2(P \wedge E = \texttt{false}) &
\end{array}
$$

We prove by "color" cases and for blue and green by induction in the size of the command.
If m' is in $sp^2_o(\gamma^2(P), if\ \ E\ \ then\ \ C_1)$, then there is a m such that :
- we are in the red case, then m' is in $\gamma^2(sp^2(P, if\ \ E\ \ then\ \ C_1))$ by his red set.
- we are in the blue case then $m' \in sp^2_o(\gamma^2(P \wedge E = \texttt{true}), C_1)$, then by induction hypothesis m' is in the blue set of $\gamma^2(sp^2(P, if\ \ E\ \ then\ \ C_1))$
- we are in the green case then $m' \in \gamma^2(P \wedge E = \texttt{false})$ and then m' is in the green set of $\gamma^2(sp^2(P, if\ \ E\ \ then\ \ C_1))$

We can conclude that $sp^2_o(\gamma^2(P), if\ \ E\ \ then\ \ C_1) \subseteq \gamma^2(sp^2(P, if\ \ E\ \ then\ \ C_1))$

C.9 *while E do C*$_1$

Lemma If $lfp\,\overset{\models^2}{P}\,\lambda X.\ Q$ and $lfp\,\overset{\subseteq}{\gamma^2(P)}\lambda X.\ \gamma^2(Q)\{\gamma(X) \to X\}$ exists then $\gamma^2(lfp\,\overset{\models^2}{P}\,\lambda X.\ Q) \supseteq lfp\,\overset{\subseteq}{\gamma^2(P)}\lambda X.\ \gamma^2(Q)\{\gamma(X) \to X\}$

with $\{A \to B\}$ the syntactical substitution

Lemma's proof : let's define $U = lfp\,\overset{\models^2}{P}\,\lambda X.\ Q$ and $F = \lambda X.\ \gamma^2(Q)\{\gamma(X) \to X\}$

U is a fixpoint of $\lambda X.\ Q$,

$$\begin{array}{lrcl} \text{so} & U & \equiv^2 & Q\{X \to U\} \\ \text{then} & \gamma^2(U) & = & \gamma^2(Q\{X \to U\}) \\ & & = & \gamma^2(Q)\{\gamma^2(X) \to \gamma^2(U)\} \\ & & = & \gamma^2(Q)\{\gamma^2(X) \to X\}\{X \to \gamma^2(U)\} \\ & & = & F(\gamma^2(U)) \end{array}$$

and so $\gamma^2(U)$ is a fixpoint of F and then $\gamma^2(U) \supseteq lfp\ F$.□

$$\begin{array}{lcl} sp^2_o(\gamma^2(P), w\ E\ d\ C_1) & = & (lfp\,\overset{\subseteq}{\gamma^2(P)}\lambda X. \quad \{m' \mid \exists m. \quad m \in X \wedge \quad (((m' = \Omega_o \wedge \quad (\ m = \Omega_o \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vee (m = s, h \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m \models^2 \neg(E = E)))) \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vee (m = s, h \quad \wedge [\![E]\!]s = True \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge m' \in sp^2_o(m, C_1)) \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vee (m' = m) \\ & & \cap\{m' \mid (m' \models^2 \quad E = \texttt{false}) \qquad\qquad\qquad\quad \vee m' = \Omega_o\} \\ \gamma^2(sp^2(P, w\ E\ d\ C_1)) & = & \gamma^2(lfp\,\overset{\models^2}{P}\,\lambda X. \qquad\qquad\qquad\qquad\quad ((\texttt{Err} \wedge \quad (X \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vee (\exists^2 m. X \wedge \neg(E = E)))) \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vee sp^2(X \wedge E = \quad \texttt{true}, C_1) \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vee X) \\ & & \wedge(E = \texttt{false} \quad \vee \texttt{Err})) \end{array}$$

So by the lemma we have that if the lfp s exist, $sp^2_o(\gamma^2(P), while\ \ E\ \ do\ \ C_1) \subseteq \gamma^2(sp^2(P, while\ \ E\ \ do\ \ C_1))$.

And if the lfp does not exist, $\gamma^2(sp^2(P, while\ \ E\ \ do\ \ C_1)) = \gamma^2(E = \texttt{false} \vee \texttt{Err})$ in which $sp^2_o(\gamma^2(P), while\ \ E\ \ do\ \ C_1)$ is included since the $\cap$ and so we have $sp^2_o(\gamma^2(P), while\ \ E\ \ do\ \ C_1) \subseteq \gamma^2(sp^2(P, while\ \ E\ \ do\ \ C_1))$.

Appendix D

Why this definition of $[x\backslash y]$

D.1 Definition 1 : recursive

$$\begin{array}{rcl} [x\backslash y] & \equiv & \neg\texttt{isloc}(x) \vee \neg\texttt{isloc}(y) \\ & & \vee \exists x_1, x_2, y_1, y_2. \\ & & \quad (x \hookrightarrow x_1, x_2) * (y \hookrightarrow y_1, y_2) \\ & & \quad \wedge [x_1\backslash y] \wedge [x_2\backslash y] \wedge [x\backslash y_1] \wedge [x\backslash y_2] \end{array}$$

This is obviously right.

But it does not allow to make proofs if we have cycles.

For example, to prove that :

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \end{array} \right], \left[\begin{array}{c} l_x \mapsto l_x, l_x \\ l_y \mapsto l_y, l_y \end{array} \right] \models [x\backslash y]$$

We should prove that *(among other things)*:

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \\ x_1 \mapsto l_x \\ x_2 \mapsto l_x \\ y_1 \mapsto l_y \\ y_2 \mapsto l_y \end{array} \right], \left[\begin{array}{c} l_x \mapsto l_x, l_x \\ l_y \mapsto l_y, l_y \end{array} \right] \models [x_1\backslash y]$$

and so on. So we would never end.

D.2 Definition 2

The idea here is that if two pointers can not reach a same location after several dereferencing, then we could split the memory into two pieces that can not reach each other and that each one have one of those pointers. Since BI have some connective $*$ that somehow split the memory, we would like to use it. But in BI we have dangling pointers (some variables in the heap are assigned to locations that are not in the heap domain), and specially $*$ do not split all the memory but only the heap and so it create dangling pointers.

So we could try to split the memory and say that there are no new dangling pointers:

$$\begin{array}{rcl} [x\backslash y] & \equiv & \neg\texttt{isloc}(x) \vee \neg\texttt{isloc}(y) \\ & & \vee \exists x_1, x_2, y_1, y_2. \\ & & \quad (x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}) * (y \hookrightarrow y_1, y_2 \wedge \texttt{Nodangling2}) \end{array}$$

But it is a wrong definition.
For example, to prove that :

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \end{array}\right], \left[\begin{array}{c} l_x \mapsto l_x, l_x \\ l_y \mapsto l_y, l_y \end{array}\right] \models [x \backslash y]$$

We should prove that *(among other things)*:

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \\ x_1 \mapsto l_x \\ x_2 \mapsto l_x \\ y_1 \mapsto l_y \\ y_2 \mapsto l_y \end{array}\right], \left[\begin{array}{c} l_x \mapsto l_x, l_x \end{array}\right] \models x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}$$

which is wrong since here we have y as a new dangling pointer, however $[x \backslash y]$ was holding.

D.3 Definition 3

Here we can see that they are two kind of dangling locations, the ones mentioned in the stack, and the ones mentioned in the heap. We call those last ones the dangling2.

For example, in :

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \end{array}\right], \left[\begin{array}{c} l_x \mapsto 1, l_z \end{array}\right]$$

l_y is a simple dangling location, while l_z is a dangling2 location.

So our third definition will be :

$$\begin{array}{lll} [x \backslash y] & \equiv & \neg\texttt{isloc}(x) \vee \neg\texttt{isloc}(y) \\ & & \vee \exists x_1, x_2, y_1, y_2. \\ & & \quad (x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}) * (y \hookrightarrow y_1, y_2 \wedge \texttt{Nodangling2}) \end{array}$$

So for example, to prove that :

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \end{array}\right], \left[\begin{array}{c} l_x \mapsto l_x, l_y \\ l_y \mapsto l_y, l_y \end{array}\right] \models [x \backslash y]$$

we will have to prove that

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \\ x_1 \mapsto l_x \\ x_2 \mapsto l_y \end{array}\right], \left[\begin{array}{c} l_x \mapsto l_x, l_y \end{array}\right] \models x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}$$

we have $x \hookrightarrow x_1, x_2$ holding but not `Nodangling2`, since l_y is a new dangling2 location, so as expected we did not had $[x \backslash y]$.

But we still have a problem .
For example, if we want to prove that

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \end{array}\right], \left[\begin{array}{c} l_x \mapsto l_x, l_z \\ l_y \mapsto l_y, l_z \end{array}\right] \models [x \backslash y]$$

we will have to prove that

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \\ x_1 \mapsto l_x \\ x_2 \mapsto l_z \end{array}\right], \left[\begin{array}{c} l_x \mapsto l_x, l_z \end{array}\right] \models x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}$$

which holds since l_z the only dangling2 pointer was already dangling

and we would also have to prove that

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \\ y_1 \mapsto l_y \\ y_2 \mapsto l_z \end{array}\right], \left[\ l_y \mapsto l_y, l_z\ \right] \models y \hookrightarrow y_1, y_2 \wedge \texttt{Nodangling2}$$

which holds for the same reason.

So we would conclude that $[x\backslash y]$ which is obviously not right.

D.4 Definition 4

So our next definition will be not to allow any dangling2.

$$\begin{array}{rcl} [x\backslash y] & \equiv & \neg\texttt{isloc}(x) \vee \neg\texttt{isloc}(y) \\ & & \vee \exists x_1, x_2, y_1, y_2. \\ & & \quad (x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}) * (y \hookrightarrow y_1, y_2 \wedge \texttt{Nodangling2}) \end{array}$$

But with this definition :

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \end{array}\right], \left[\begin{array}{c} l_x \mapsto 1, 2 \\ l_y \mapsto 3, 4 l_z \mapsto l_w, l_w \end{array}\right] \models \neg[x\backslash y]$$

$[x\backslash y]$ will not hold since to prove it we will either have to prove that :

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \\ x_1 \mapsto 1 \\ x_2 \mapsto 2 \end{array}\right], \left[\ l_x \mapsto 1, 2 l_z \mapsto l_w, l_w\ \right] \models x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}$$

which does not hold because of l_w
or either have to prove that:

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \\ y_1 \mapsto 3 \\ y_2 \mapsto 4 \end{array}\right], \left[\ l_y \mapsto 3, 4 l_z \mapsto l_w, l_w\ \right] \models x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}$$

which does not hold because of l_w.
where we could want to say that :

$$\left[\begin{array}{c} x \mapsto l_x \\ y \mapsto l_y \end{array}\right], \left[\begin{array}{c} l_x \mapsto 1, 2 \\ l_y \mapsto 3, 4 l_z \mapsto l_w, l_w \end{array}\right] \models [x\backslash y]$$

D.5 Definition 5

So our definition will split the memory (heap), into 3 piece, two for x and y with no dangling2 and one for the rest.

$$\begin{array}{rcl} [x\backslash y] & \equiv & \neg\texttt{isloc}(x) \vee \neg\texttt{isloc}(y) \\ & & \vee \exists x_1, x_2, y_1, y_2. \\ & & \quad (x \hookrightarrow x_1, x_2 \wedge \texttt{Nodangling2}) * (y \hookrightarrow y_1, y_2 \wedge \texttt{Nodangling2}) * \texttt{Nodangling2} \end{array}$$

And it will be ok for the previous example.

Remark : we need the `Nodangling2` to hold also for the "rest" of the heap since otherwise, x and y could not be able to reach each other but some other pointer might be able to reach both of them.

D.6 A step to γ_P

Now that we got a definition of $[x\backslash y]$ that seems to works *(it would be great to prove that the 5th definition implies the first one, but with the way the first one is written, we have nothing decreasing and so we can not prove anything)*, we have to define γ_P.

Since P is a set of $[a\backslash b]$, the first thing we thought to do was just,

$$\gamma_P(P) = \bigwedge_{[a\backslash b]\in P} [a\backslash b]$$

.

But if we look at $sp_P(P, x = new(x))$, we have $\forall z.\ [x\backslash z]$, which can obviously not be right with our definition. For example if before the execution of $x = new(x)$, we had y reaching a dangling2 pointer, the execution will not have changed it, and y will still be reaching a dangling2 pointer, and we would not have $[x\backslash y]$.

So we have change the γ_P into:

$$\gamma_p(P) \quad = \quad \bigwedge_{[a\backslash b]\in P} [a\backslash b] \wedge \texttt{Nodangling2} \wedge \bigwedge_{\forall z\text{“}\in\text{''}P} \neg\, \texttt{isdangling}(z)$$

which says that we only work on memory with no dangling and dangling2 pointers, which is conserved after all our commands executions.

Appendix E

Substitutions formulas

E.1 $[\![E\{E'/x\}]\!]s = [\![E]\!]s[x \mapsto [\![E']\!]s]$ if $[\![E']\!]s$ exists

- $[\![x\{E'/x\}]\!]s = [\![E']\!]s = [\![x]\!]s[x \mapsto [\![E']\!]s]$
- $[\![y\{E'/x\}]\!] = [\![y]\!]s = [\![y]\!]s[x \mapsto [\![E']\!]s]$
- $[\![\texttt{true}\{E'/x\}]\!] = [\![\texttt{true}]\!]s = true = [\![\texttt{true}]\!]s[x \mapsto [\![E']\!]s]$
- $[\![\texttt{false}\{E'/x\}]\!] = [\![\texttt{false}]\!]s = false = [\![\texttt{false}]\!]s[x \mapsto [\![E']\!]s]$
- $[\![42\{E'/x\}]\!] = [\![42]\!]s = 42 = [\![42]\!]s[x \mapsto [\![E']\!]s]$
- $[\![(E_1 \ op \ E_2)\{E'/x\}]\!] = [\![E_1\{E'/x\} \ op \ E_2\{E'/x\}]\!]s = [\![E_1\{E'/x\}]\!]s \quad op \quad [\![E_2\{E'/x\}]\!]s = [\![E_1]\!]s[x \mapsto [\![E']\!]s] \ op \ [\![E_2]\!]s[x \mapsto [\![E']\!]s] = [\![E_1 \ op \ E_2]\!]s[x \mapsto [\![E']\!]s]$

E.2 Proof that $P[E/x] \equiv P\{E/x\}$

Remember $is(E') \equiv E' = E'$ so $s, h \models is(E')$ iff $[\![E']\!]s$ exists.

In this section we prove that the formula $P[E'/x]$ is not add to the BI syntax and it is just a shortcut for $P\{E'/x\} \wedge is(E')$.

Recall that $s, h \models P[E'/x]$ iff $s[x \mapsto [\![E']\!]s], h \models P$.

By induction on the size of the proposition P:

- $s, h \models (E_1 = E_2)\{E'/x\} \wedge is(E')$
  iff $s, h \models E_1\{E'/x\} = E_2\{E'/x\} \wedge is(E')$
  iff $[\![E_1\{E'/x\}]\!]s = [\![E_2\{E'/x\}]\!]s$ and $[\![E']\!]s$ exists
  iff $[\![E_1]\!]s[x \mapsto [\![E']\!]s] = [\![E_2]\!]s[x \mapsto [\![E']\!]s]$
  iff $s[x \mapsto [\![E']\!]s], h \models E_1 = E_2$
  iff $s, h \models (E_1 = E_2)[E'/x]$

- $s, h \models (E \mapsto E_1, E_2)\{E'/x\} \wedge is(E')$
  iff $s, h \models E\{E'/x\} \mapsto E_1\{E'/x\}, E_2\{E'/x\} \wedge is(E')$
  iff $dom(h) = \{[\![E\{E'/x\}]\!]s\}$ and $h([\![E\{E'/x\}]\!]s) = \langle [\![E_1\{E'/x\}]\!]s, [\![E_2\{E'/x\}]\!]s \rangle$
  and $[\![E']\!]s$ exists
  iff $dom(h) = \{[\![E]\!]s[x \mapsto [\![E']\!]s]\}$
  and $h([\![E]\!]s[x \mapsto [\![E']\!]s]) = \langle [\![E_1]\!]s[x \mapsto [\![E']\!]s], [\![E_2]\!]s[x \mapsto [\![E']\!]s] \rangle$
  iff $s[x \mapsto [\![E']\!]s], h \models E \mapsto E_1, E_2$
  iff $s, h \models (E \mapsto E_1, E_2)[E'/x]$

- $s,h \models \texttt{false}\{E'/x\} \wedge is(E')$
  iff $s,h \models \texttt{false} \wedge is(E')$
  iff $s[x \mapsto [\![E']\!]s], h \models \texttt{false}$
  iff $s,h \models \texttt{false}[E'/x]$

- $s,h \models (P \Rightarrow Q)\{E'/x\} \wedge is(E')$
  iff $s,h \models (P\{E'/x\} \Rightarrow Q\{E'/x\}) \wedge is(E')$
  iff $[\![E']\!]s$ exists and If $s,h \models P\{E'/x\}$ then $s,h \models Q\{E'/x\}$
  iff $[\![E']\!]s$ exists and If $s[x \mapsto [\![E']\!]s], h \models P$ then $s[x \mapsto [\![E']\!]s], h \models Q$
  iff $s[x \mapsto [\![E']\!]s], h \models P \Rightarrow Q$
  iff $s,h \models (P \Rightarrow Q)[E'/x]$

- $s,h \models (\exists x.\ P)\{E'/x\} \wedge is(E')$
  iff $s,h \models (\exists x.\ P) \wedge is(E')$
  iff $\exists v. s[x \mapsto v], h \models P$ and $[\![E']\!]s$ exists
  iff $\exists v. s[x \mapsto [\![E']\!]s][x \mapsto v], h \models P$ and $[\![E']\!]s$ exists
  iff $s[x \mapsto [\![E']\!]s], h \models \exists x.\ P$
  iff $s,h \models (\exists x.\ P)[E'/x]$

- $s,h \models (\exists y.\ P)\{E'/x\} \wedge is(E')$ with $x \neq y$
  iff $s,h \models (\exists y.\ P\{E'/x\}) \wedge is(E')$
  iff $\exists v. s[y \mapsto v], h \models P\{E'/x\}$ and $[\![E']\!]s$ exists
  iff $\exists v. s[y \mapsto v][x \mapsto [\![E']\!]s], h \models P$ and $[\![E']\!]s$ exists
  iff $\exists v. s[x \mapsto [\![E']\!]s][y \mapsto v], h \models P$ and $[\![E']\!]s$ exists
  iff $s[x \mapsto [\![E']\!]s], h \models \exists y.\ P$
  iff $s,h \models (\exists y.\ P)[E'/x]$

- $s,h \models \texttt{emp}\{E'/x\} \wedge is(E')$
  iff $s,h \models \texttt{emp} \wedge is(E')$
  iff $s[x \mapsto [\![E']\!]s], h \models \texttt{emp}$
  iff $s,h \models \texttt{emp}[E'/x]$

- $s,h \models (P * Q)\{E'/x\} \wedge is(E')$
  iff $s,h \models (P\{E'/x\} * Q\{E'/x\}) \wedge is(E')$
  iff $\exists h_0, h_1.\ h_0 \sharp h_1, h_0.h_1 = h, s, h_0 \models P\{E'/x\}$ and $s, h_1 \models Q\{E'/x\}$ and $[\![E']\!]s$ exists
  iff $\exists h_0, h_1.\ h_0 \sharp h_1, h_0.h_1 = h, s[x \mapsto [\![E']\!]s], h_0 \models P$ and $s[x \mapsto [\![E']\!]s], h_1 \models Q$ and $[\![E']\!]s$ exists
  iff $s[x \mapsto [\![E']\!]s], h \models (P * Q)$
  iff $s,h \models (P * Q)[E'/x]$

- $s,h \models (P\{E'/x\} \mathrel{-\!\!*} Q\{E'/x\}) \wedge is(E')$
  iff $\forall h'.$ if $h' \sharp h$ and $s,h' \models P\{E'/x\}$ then $s, h.h' \models Q\{E'/x\}$ and $[\![E']\!]s$ exists
  iff $\forall h'.$ if $h' \sharp h$ and $s[x \mapsto [\![E']\!]s], h' \models P$ then $s[x \mapsto [\![E']\!]s], h.h' \models Q$ and $[\![E']\!]s$ exists
  iff $s[x \mapsto [\![E']\!]s], h \models P \mathrel{-\!\!*} Q$
  iff $s,h \models (P \mathrel{-\!\!*} Q)[E'/x]$

E.3 Proof that $y = E[E'/x] \equiv y = (E\{E'/x\}) \wedge is(E')$

In this section we prove that the formula $y = E[E'/x]$ is not add to the BI syntax and it is just a shortcut for $y = (E\{E'/x\}) \wedge is(E')$.

Recall that $s,h \models y = E[E'/x]$ iff $s(y) = [\![E]\!]s[x \mapsto [\![E']\!]s]$.

- $s,h \models y = E\{E'/x\} \wedge is(E')$
  iff $[\![y]\!]s = [\![E\{E'/x\}]\!]s$ and $[\![E']\!]s$ exists
  iff $s(y) = [\![E\{E'/x\}]\!]s$ and $[\![E']\!]s$ exists
  iff $s(y) = [\![E]\!]s[x \mapsto [\![E']\!]s]$ by E.1
  iff $s,h \models y = E[E'/x]$

E.4 Proof that $y = E[E'/x].i \equiv y = (E\{E'/x\}).i \wedge is(E')$

In this section we prove that the formula $y = E[E'/x].i$ is not add to the BI syntax and it is just a shortcut for $y = (E\{E'/x\}).i \wedge is(E')$.

Actually, $y = E.i$ is not a BI's formula, but we write it as a shortcut for the formula $\exists x_1, x_2.\ (E \hookrightarrow x_1, x_2) \wedge (y = x_i)$.

Recall that $s, h \models y = E[E'/x]$ iff $s(y) = \pi_i(h([\![E]\!]s[x \mapsto [\![E_2]\!]s]))$.

- $s, h \models y = (E\{E'/x\}).i \wedge is(E')$
  iff $s, h \models \exists x_1, x_2.\ ((E\{E'/x\}) \hookrightarrow x_1, x_2) \wedge (y = x_i) \wedge is(E')$
  iff $\exists v_1, v_2.\ s[x_1 \mapsto v_1][x_2 \mapsto v_2], h \models ((E\{E'/x\}) \hookrightarrow x_1, x_2) \wedge (y = x_i)$ and $[\![E']\!]s$ exists
  iff $\exists v_1, v_2.\ s[x_1 \mapsto v_1][x_2 \mapsto v_2], h \models ((E\{E'/x\}) \hookrightarrow x_1, x_2)$ and $s[x_1 \mapsto v_1][x_2 \mapsto v_2], h \models y = x_i$ and $[\![E']\!]s$ exists
  iff $\exists v_1, v_2.\ s[x_1 \mapsto v_1][x_2 \mapsto v_2], h \models (E \hookrightarrow x_1, x_2)\{E'/x\})$ and $s(y) = v_i$ and $[\![E']\!]s$ exists
  iff $\exists v_1, v_2.\ s[x_1 \mapsto v_1][x_2 \mapsto v_2][x \mapsto [\![E']\!]s], h \models (E \hookrightarrow x_1, x_2))$ and $s(y) = v_i$, by E.2
  iff $\exists v_1, v_2.\ h([\![E]\!]s[x_1 \mapsto v_1][x_2 \mapsto v_2][x \mapsto [\![E']\!]s]) = \langle v_1, v_2\rangle$ and $s(y) = v_i$
  iff $\exists v_1, v_2.\ h([\![E]\!]s[x \mapsto [\![E']\!]s]) = \langle v_1, v_2\rangle$ and $s(y) = v_i$
  iff $s(y) = \pi_i(h([\![E]\!]s[x \mapsto [\![E']\!]s]))$
  iff $s, h \models y = E[E'/x].i$

E.5 Proof that $y \mapsto E_1[E'/x], E_2[E'/x] \equiv y \mapsto E_1\{E'/x\}, E_2\{E'/x\} \wedge is(E')$

In this section we prove that the formula $y \mapsto E_1[E'/x], E_2[E'/x]$ is not add to the BI syntax and it is just a shortcut for $y \mapsto E_1\{E'/x\}, E_2\{E'/x\} \wedge is(E')$.

Recall that $s, h \models y \mapsto E_1[E'/x], E_2[E'/x]$ iff $dom(h) = \{s(y)\}$ and $h(s(y)) = \langle [\![E_1]\!]s[x \mapsto [\![E']\!]s], [\![E_2]\!]s[x \mapsto [\![E']\!]s]\rangle$.

- $s, h \models y \mapsto E_1\{E'/x\}, E_2\{E'/x\} \wedge is(E')$
  iff $dom(h) = \{s(y)\}$ and $h(s(y)) = \langle [\![E_1\{E'/x\}]\!]s, [\![E_2\{E'/x\}]\!]s\rangle$ and $[\![E']\!]s$ exists
  iff $dom(h) = \{s(y)\}$ and $h(s(y)) = \langle [\![E_1]\!]s[x \mapsto [\![E']\!]s], [\![E_2]\!]s[x \mapsto [\![E']\!]s]\rangle$, by E.1
  iff $s, h \models y \mapsto E_1[E'/x], E_2[E'/x]$

Bibliography

[1] Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. *Lecture Notes in Computer Science*, 2245:108–??, 2001.

[2] P. Cousot and R. Cousot. Static determination of dynamic properties of generalized type unions. In *ACM Symposium on Language Design for Reliable Software*, Raleigh, North Calorina, ACM SIGPLAN Notices 12(3):77–94, 1977.

[3] D. Galmiche and D. Méry. Connection-based proof search in propositional BI logic. In *Int. Conference on Automated Deduction, CADE'02*, Copenhagen, Danemark, July 2002.

[4] Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *POPL'01*, pages 14–26, 2001.

[5] P. W. O'Hearn and D. J. Pym. The logic of bunched implications. In *Bulletin of Symbolic Logic*, June 1999.

[6] Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In LNCS 2142 ©Springer-Verlag, editor, *Proceedings of CSL'01*, pages 1–19, Paris, 2001.

[7] John Reynolds. Separation logic : A logic for shared mutable data structures. In *LICS'02*, Copenhagen, Denmark, July 22-25 2002.

[8] Hongseok Yang and Peter W. O'Hearn. A semantic basis for local reasoning. In *Proceedings of FOSSACS'02*, 2002.