An abstract domain for separation logic formulae

Élodie-Jane Sims
Elodie-Jane.Sims@polytechnique.fr

Polytechnique/KSU | LIAFA - Séminaire Vérification | 19 juin 2006
Séminaire au CEA | 22 juin 2006

Plan

- Goal: pointer analysis: check dereferencing errors, aliases, ...
- $BI^{\mu\nu}$ a separation logic which permit easy descriptions of the memory, e.g.
  - x points to a list of [1;2;3]
    $\exists x_2, x_3.\ (x \hookrightarrow 1, x_2) * (x_2 \hookrightarrow 2, x_3) * (x_3 \hookrightarrow 3, \texttt{nil})$
  - x and y are aliased pointers
    $x = y \wedge \exists x_1, x_2.\ (x \hookrightarrow x_1, x_2)$
  - Partitionning: x and y belong to two disjoints parts of the heap which have no pointers from one to the other...

Example of a pointer program with a bug

$$\uparrow$$

$$\{(\exists z_1, z_2.\ \mathtt{nil} \hookrightarrow z_1, z_2) \equiv \boldsymbol{FALSE}\}$$

$$\boldsymbol{x :\ = \mathtt{nil};}$$

$$\{\exists z_1, z_2.\ x \hookrightarrow z_1, z_2\}$$

$$\boldsymbol{z :\ = x;}$$

$$\{\exists z_1, z_2.\ z \hookrightarrow z_1, z_2\}$$

$$\boldsymbol{y :\ = z \cdot 1;}$$

$$\{\boldsymbol{TRUE}\}$$

Point-to analyses: Shape/alias analyses

- Shape analyses: the analysis build a graph where
  - the nodes represent locations in the heap
  - the edges represent fields between locations

  The analysis usually do approximation (represent more or less nodes/fields than what is in the heap) and computes some more informations about the nodes or edges of the graph.

  Recent examples : TVLA (Sagiv, Reps, Wilhelm,...), Smallfoot (O'Hearn, Yang, Berdine, Calcagno, Distefano,...)

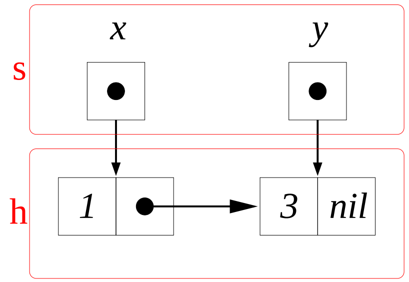
- Alias analyses: a point-to analysis which computes sets of variables

- Goal: pointer analysis: check dereferencing errors, aliases, ...

- $BI^{\mu\nu}$ a separation logic which permit easy descriptions of the memory, e.g.

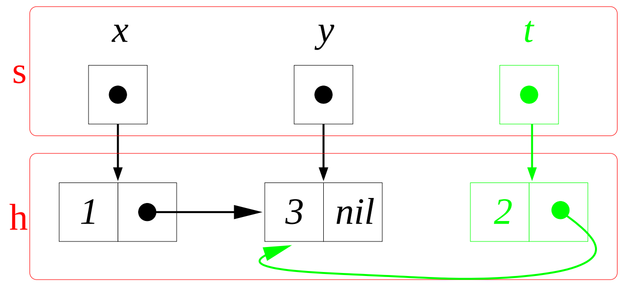
  – x points to a list of [1;2;3]
    $\exists x_2, x_3.\ (x \hookrightarrow 1, x_2) * (x_2 \hookrightarrow 2, x_3) * (x_3 \hookrightarrow 3, \texttt{nil})$
  – x and y are aliased pointers
    $x = y \wedge \exists x_1, x_2.\ (x \hookrightarrow x_1, x_2)$
  – Partitionning: x and y belong to two disjoints parts of the heap which have no pointers from one to the other...

Example for a piece of code inserting a cell in a linked list



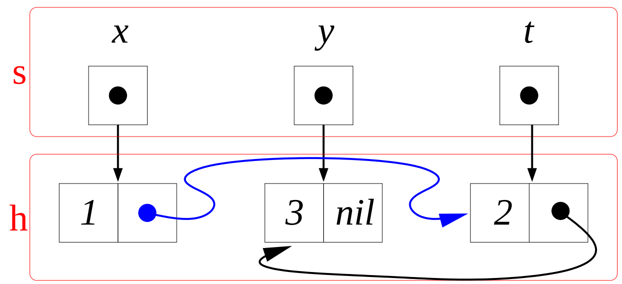
$$\{(x \mapsto 1, y) * (y \mapsto 3, \mathtt{nil})\}$$

$$\boldsymbol{t :\ = \mathtt{cons}(2, y)};$$



$$\{(x \mapsto 1, y) * (y \mapsto 3, \mathtt{nil}) \boldsymbol{* (t \mapsto 2, y)}\}$$

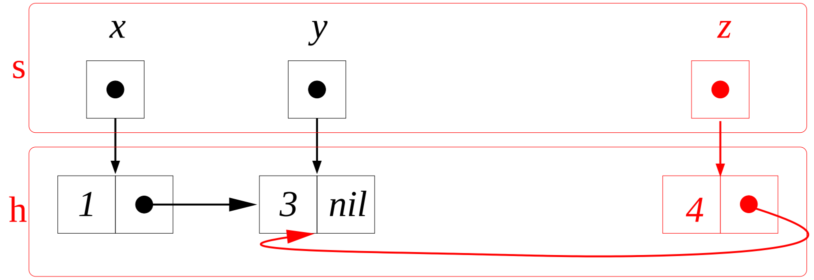
$$\boldsymbol{x \cdot 2 :\ = t};$$



$$\{(x \mapsto 1, \boldsymbol{t}) * (t \mapsto 2, y) * (y \mapsto 3, \mathtt{nil})\}$$

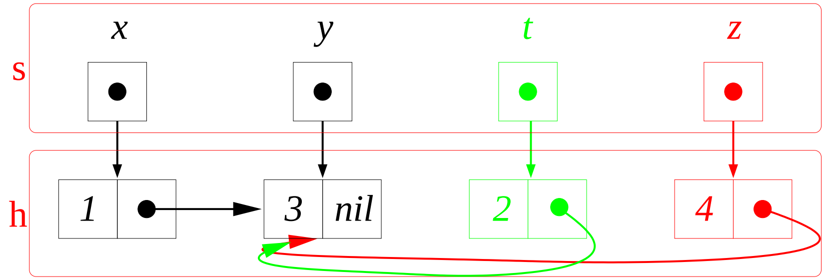
Local reasoning

$$\left\{ \begin{array}{l} (x \mapsto 1, y) * (y \mapsto 3, \mathtt{nil}) \\ *(z \mapsto 4, y) \end{array} \right\}$$



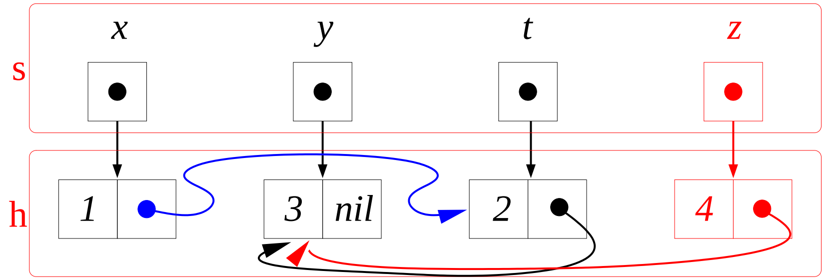
$$t := \mathtt{cons}(2, y);$$

$$\left\{ \begin{array}{l} (x \mapsto 1, y) * (y \mapsto 3, \mathtt{nil}) * (t \mapsto 2, y) \\ *(z \mapsto 4, y) \end{array} \right\}$$

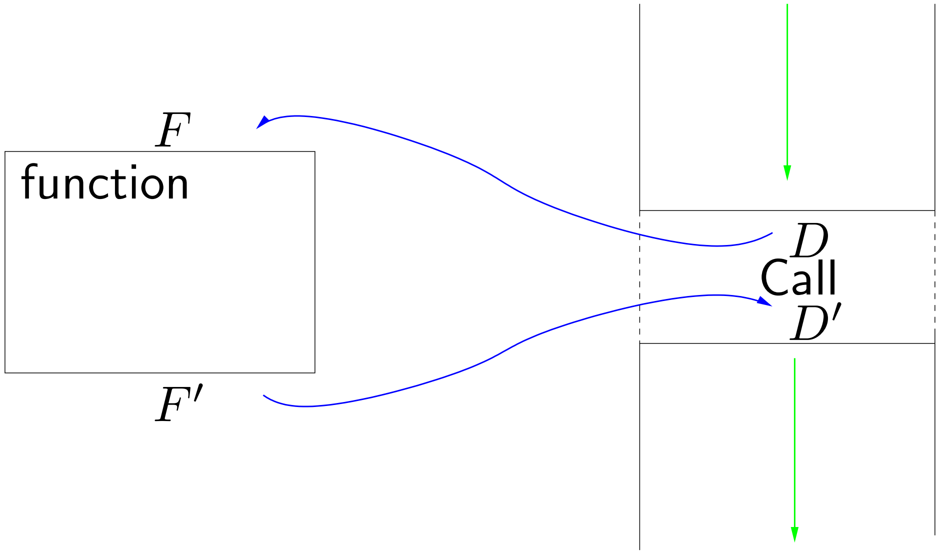


$$x \cdot 2 := t;$$

$$\left\{ \begin{array}{l} (x \mapsto 1, t) * (t \mapsto 2, y) * (y \mapsto 3, \mathtt{nil}) \\ *(z \mapsto 4, y) \end{array} \right\}$$



- We want to use this logic as an interface language for modular analysis



Analysis 1 $\longrightarrow BI^{\mu\nu} \longrightarrow$ Analysis 2
Program $\longrightarrow BI^{\mu\nu} \longrightarrow$ Analysis 3

We have build an intermediate domain such that:

- it is similar to the existing shape/alias analysis domains to allow translations from/to those domains
- it comes with a concrete semantics in term of sets of states which is the same domain as for the formulae's semantics
- we can translate the formulas into our domain
- it is a cartesian product of different subdomains so that we can cheaply tune the precision depending on the needs (for example, the domain is parametrised by a numerical domain which can be forgotten if we do not care about numericals)

Plan

Domaine of interpretation: $State$

We have a set of variables Var.

$$\begin{array}{rcll} Val & = & Int \cup Bool \cup Atoms \cup Loc & Values \\ S & = & Var \rightharpoonup Val & Stacks \\ H & = & Loc \rightharpoonup Val \times Val & Heaps \\ State & = & S \times H & \end{array}$$

Rq: stacks can be partial functions

The logic: $BI^{\mu\nu}$

	Classical	*connectives*			
	$E = E'$		|	`false`	
|	$P \Rightarrow Q$		|	$\exists x.P$	
	Spatial	*connectives*			
|	`emp`	Empty heap	|	$E \mapsto E_1, E_2$	Points to
|	$P * Q$	Spatial conj.	|	$P -\!\!* Q$	Spatial imp.
	Fixpoints	*connectives*			
|	X_v	Variable for formulae	|	$P[E/x]$	Posponned substitution
|	$\nu X_v.P$	Greatest fixpoint	|	$\mu X_v.P$	Least fixpoint

$Var_v = \{X_v, Y_v, ...\}$ infinite set of variables of formulae

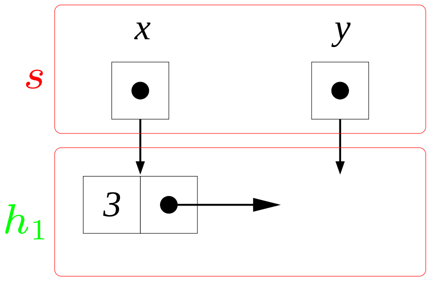
Semantic of $*$

$$[\![P * Q]\!]_\rho = \left\{ s, h_0 \cdot h_1 \;\middle|\; \begin{array}{ll} \bullet & \boldsymbol{dom(h_0) \cap dom(h_1) = \emptyset} \\ \bullet & s, h_0 \in [\![P]\!]_\rho \\ \bullet & s, h_1 \in [\![Q]\!]_\rho \end{array} \right\}$$

Examples of formulae

Ex. 1

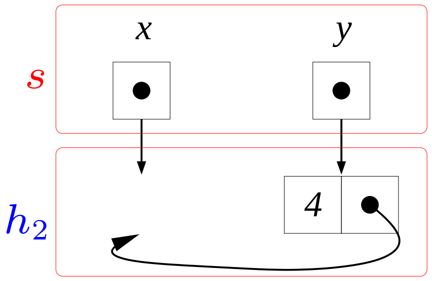
$$s = [x \longrightarrow l_1, y \longrightarrow l_2]$$
$$h_1 = [l_1 \longrightarrow \langle 3, l_2 \rangle]$$



$$\models \quad (x \mapsto 3, y)$$

Ex. 2

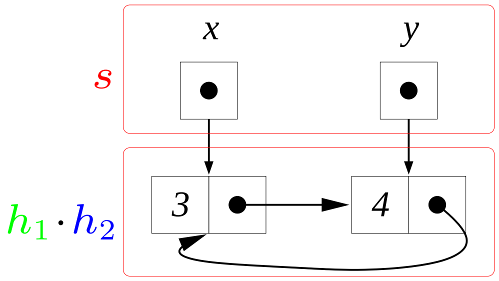
$$s = [x \longrightarrow l_1, y \longrightarrow l_2]$$
$$h_2 = [l_2 \longrightarrow \langle 4, l_1 \rangle]$$



$$\models (y \mapsto 4, x)$$

Ex. 3

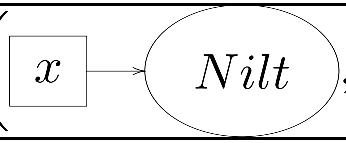
$$s = [x \longrightarrow l_1, y \longrightarrow l_2]$$
$$h_1 \cdot h_2 = \begin{bmatrix} l_1 \longrightarrow \langle 3, l_2 \rangle, \\ l_2 \longrightarrow \langle 4, l_1 \rangle \end{bmatrix}$$

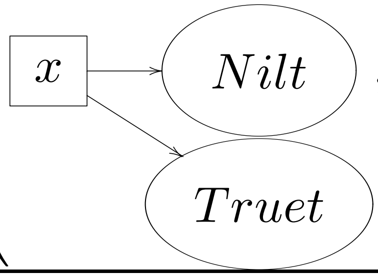


$$\models (x \mapsto 3, y) * (y \mapsto 4, x)$$
$$\not\models (x \mapsto 3, y) \wedge (y \mapsto 4, x)$$

Plan

Ex1

Formulae	$x = \texttt{nil}$
Semantics	$\{s, h \mid s(x) = \texttt{nil}\}$,...
Translation	$\Big($  $, \_, \_, \_, \_, \_, \_ \Big)$

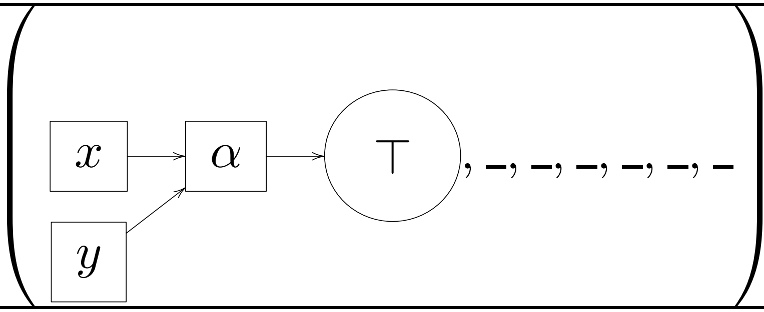
Formulae	$(x = \texttt{nil} \vee x = \texttt{true})$
Translation	$\Big($  $, \_, \_, \_, \_, \_, \_ \Big)$

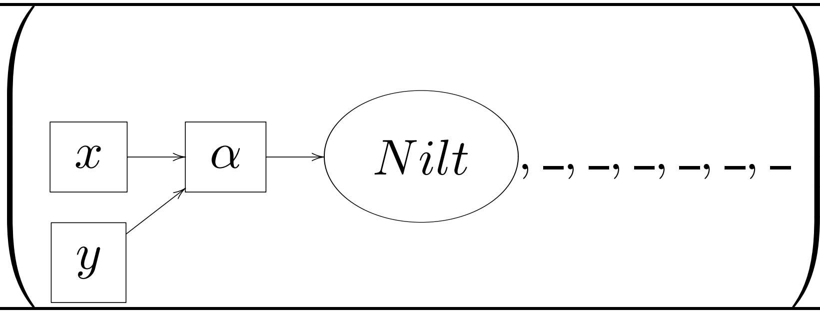
Ex2

Formula	$A \wedge B$
Constraints	cheap translation of $\wedge$
Translation	$T(A \wedge B) \triangleq T'(T'(\top, A), B)$

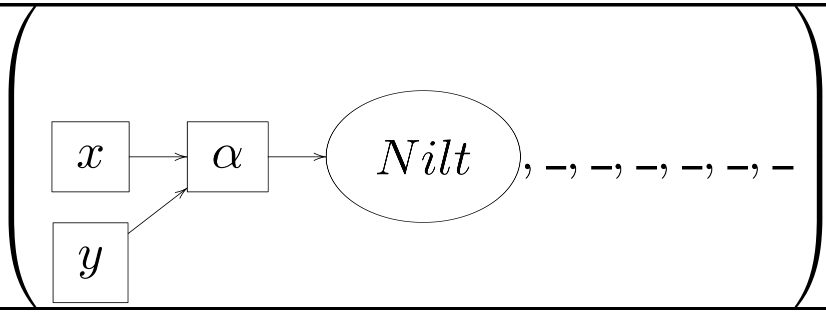
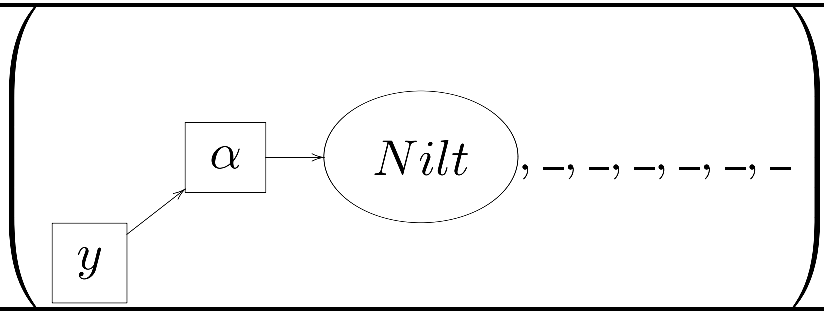
$\top$ is the empty graph, representing no information

Ex3, Ex4

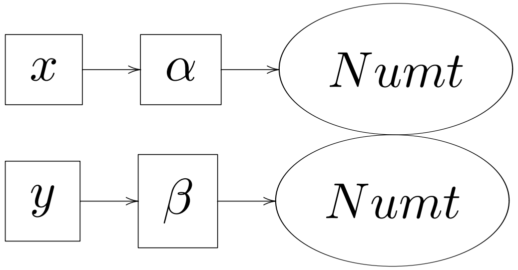
Formula	$x = y$
Constraints	refine the information for one variables while also refining the information of the second one in a cheap way
Adds	infinite set of auxiliary variables $TVar$ $VAR \triangleq Var \uplus TVar$
Translation	

Formula	$x = y \wedge x = \texttt{nil}$
Translation	

Ex5

Formula	$x = y \wedge x = \mathtt{nil}$
Translation	
Formula	$(\exists x.\ x = y \wedge x = \mathtt{nil}) \equiv (y = \mathtt{nil})$
Translation	

Ex6

Formula	$(x < y + 3)$
Translation	$\Big($  $, \_, \_, \_, \_, \_, d \Big)$ $d \in \mathcal{D}$ encodes that $\alpha < \beta + 3$

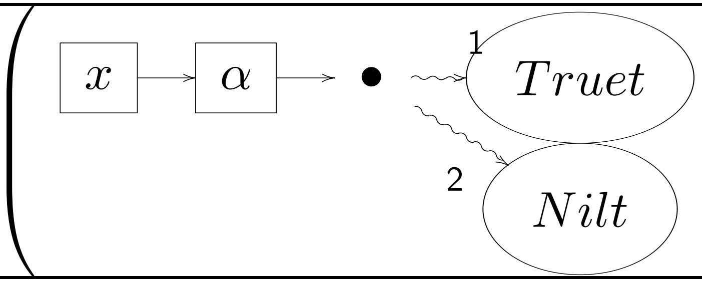
Ex7

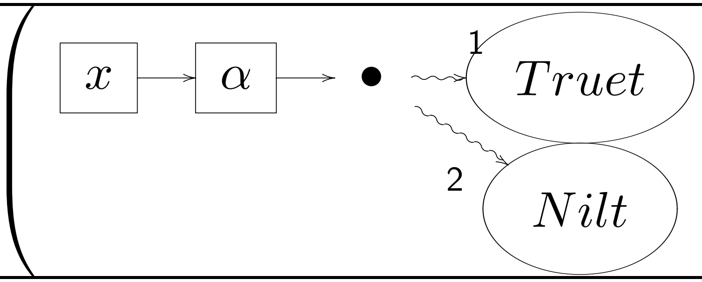
Formula	"x is a location not allocated"
Semantic	$\{s, h \mid s(x) \in Loc \wedge s(x) \notin dom(h)\}$
Translation	( $, \_, \_, \_, \_, \_, \_$)

Ex8

Formula	$\texttt{emp}$
Semantic	$\{s, h \mid dom(h) = \emptyset\}$
Adds	$HU \triangleq \mathcal{P}(TVar)$ $HO \triangleq \mathcal{P}(TVar) \uplus \texttt{full}$
Translation	$(\top, \_, \emptyset, \_, \_, \_, \_)$

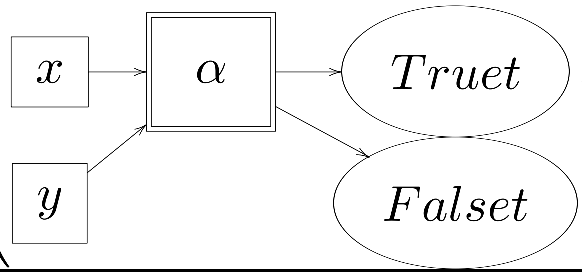
Ex9

Formula	$(x \mapsto \texttt{true}, \texttt{nil})$
Semantic	$\{s,h \mid [s(x) \rightarrow \langle True, \texttt{nil} \rangle] = h\}$
Translation	 $, \{\alpha\}, \{\alpha\}, \_, \_, \_, \_$

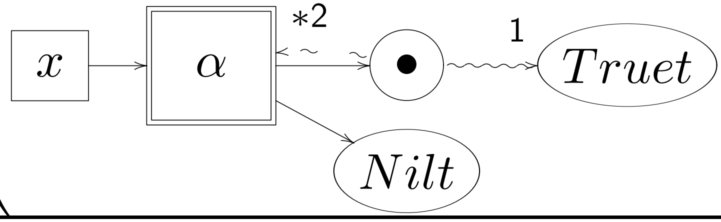
Formula	$(x \hookrightarrow \texttt{true}, \texttt{nil})$
Semantic	$\{s,h \mid [s(x) \rightarrow \langle True, \texttt{nil} \rangle] \subseteq h\}$
Translation	 $, \{\alpha\}, \texttt{full}, \_, \_, \_, \_$

Ex10

Variables represent at most one value. To allow approximation we introduce summary nodes which can represent several values.

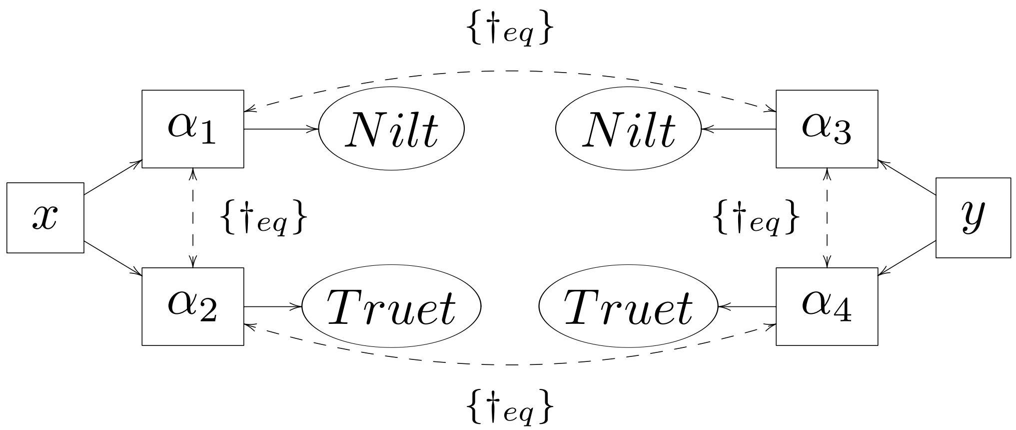
Formula	approx. of $(x = \texttt{true} \wedge y = \texttt{false})$ by $\begin{pmatrix} x = \texttt{true} \\ \vee x = \texttt{false} \end{pmatrix} \wedge \begin{pmatrix} y = \texttt{false} \\ \vee y = \texttt{true} \end{pmatrix}$
Translation	$\Bigg($  $, \_, \_, \{\alpha\}, \_, \_, \_ \Bigg)$

Ex11: finite acyclic list of $True$ starting from x

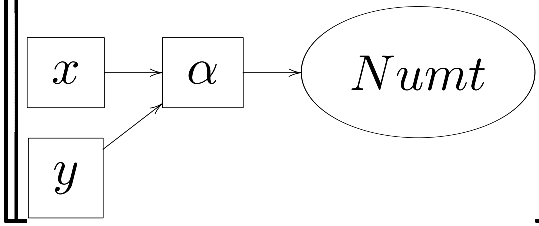
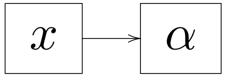
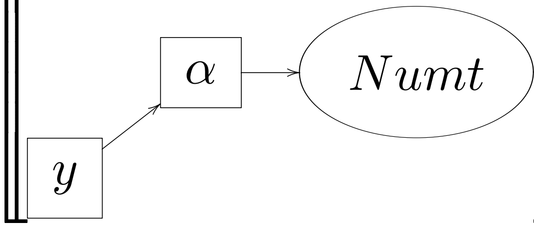
Formula	$\mu X_v.\left(\begin{array}{c}(x = \texttt{nil}) \vee \exists x_2. \\ x \hookrightarrow (\texttt{true}, x_2) \ * \ X_v[x_2/x]\end{array}\right)$
Translation	$\Big($  $, \_, \_, \{\alpha\}, \emptyset, \_, \_ \Big)$

$\emptyset$ is the set of infinite summary nodes, for infinite list μ would be replaced by ν and $\emptyset$ by $\{\alpha\}$).

Ex12: increase precision of union

<table>
<tr><td>Formula</td><td>$\begin{pmatrix} x = \texttt{nil} \\ \wedge y = \texttt{true} \end{pmatrix} \vee \begin{pmatrix} x = \texttt{true} \\ \wedge y = \texttt{nil} \end{pmatrix}$</td></tr>
<tr><td>Translation</td><td>
</td></tr>
</table>

Plan

We want $\left[\!\left[\right.\right.$  $\left.\left. \right]\!\right]^4 = \left[\!\left[\right.\right.$  $\left.\left. \right]\!\right]^4 \cap \left[\!\left[\right.\right.$  $\left.\left. \right]\!\right]^4$.

Semantic

$$
\begin{array}{rcl}
Val & \triangleq & \mathbb{Z} \uplus Bool \uplus \texttt{nil} \uplus Loc \\
S & \triangleq & Var \rightharpoonup Val \\
H & \triangleq & Loc \rightharpoonup (Val \times Val) \\
 & & \\
State & \triangleq & S \times H
\end{array}
\qquad
\begin{array}{rcl}
Val' & \triangleq & Val \cup \{\texttt{ood}\} \\
S' & \triangleq & Var \stackrel{total}{\rightarrow} Val' \\
F & \triangleq & TVar \stackrel{total}{\rightarrow} \mathcal{P}(Val') \\
R & \triangleq & Loc \rightharpoonup \mathcal{P}(Loc) \\
MFR & \triangleq & \mathcal{P}(S' \times H \times F \times R)
\end{array}
$$

$$
\begin{array}{rcl}
[\![\cdot]\!] & \in & \boldsymbol{AR} \rightarrow \boldsymbol{\mathcal{P}(State)} \\
[\![ar]\!] & \triangleq & \{\bar{s}, h \mid s, h, f, r \in [\![ar]\!]'\}
\end{array}
$$

$$
\begin{array}{rcl}
[\![\cdot]\!]' & \in & \boldsymbol{AR} \rightarrow \boldsymbol{MFR} \\
[\![(ad, hu, ho, sn, sn^{\infty}, t, d)]\!]' & \triangleq & [\![ad]\!]^4 \cap [\![hu]\!]^1 \cap [\![ho]\!]^{1'} \cap [\![sn]\!]^2 \cap [\![sn^{\infty}]\!]^{2'} \\
& & \cap [\![t]\!]^3 \cap [\![d]\!]^7 \cap sem*
\end{array}
$$

$$
\begin{array}{rcl}
[\![\cdot]\!]^4 & \in & \boldsymbol{AD} \rightarrow \boldsymbol{MFR} \\
[\![ad]\!]^4 & \triangleq & \bigcap_{v \in VAR} [\![v, ad(v)]\!]^5
\end{array}
$$

Operations

– union, intersection

– extension (replace $[v \longrightarrow S]$ by $[v \longrightarrow \{v'\}|v' \longrightarrow S]$ with a fresh v')
used to tune the precision of the union

– merging (replace $[v_1 \longrightarrow S_1 \mid v_2 \longrightarrow S_2]$ by $[v_2 \longrightarrow (S_1 \cup S_2)]$)
used with the widening

– translations from formulae to the domain

Plan

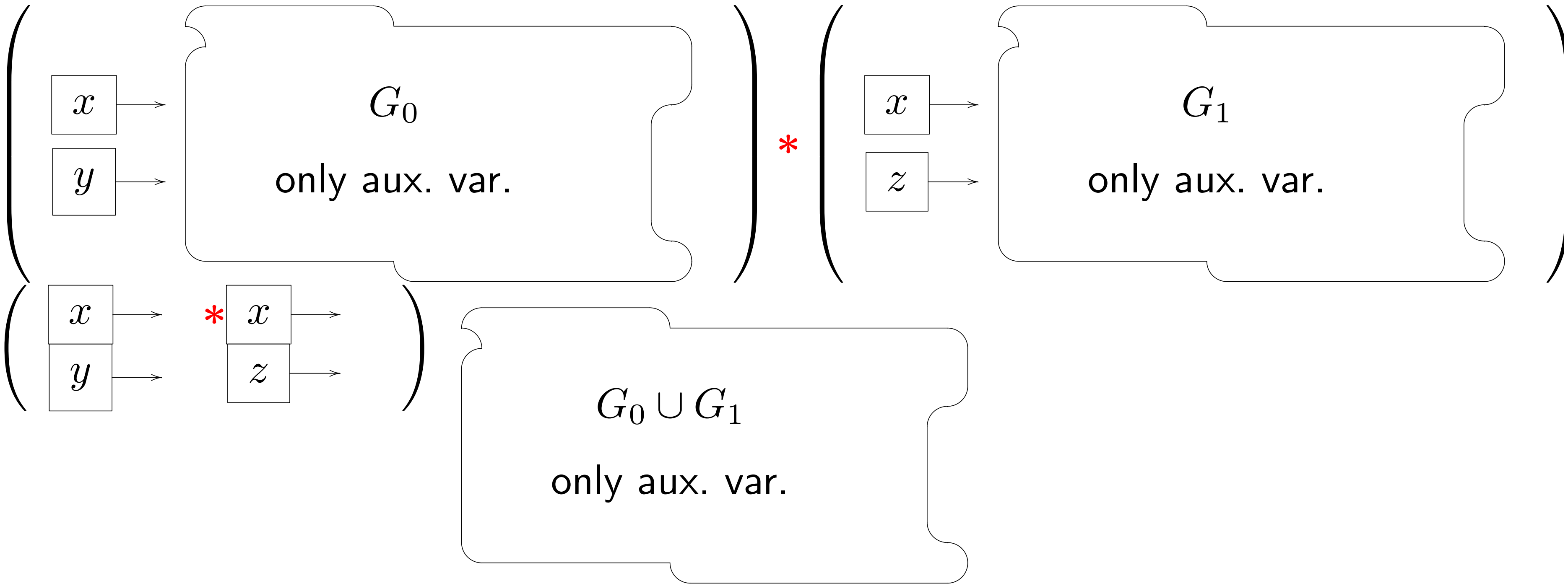
$*$ for simple types (no variables)

When $A, B \in \{Nilt, Truet, Falset, Oodt, Numt\}$, $A \neq B$

$*$	$\{A\}$	$\{B\}$	$\{Dgt\}$	$\{Loct\}$	$\{Loc(...)\}$	$\top$
$\{A\}$	$\{A\}$	Ω	Ω	Ω	Ω	$\{A\}$
$\{Dgt\}$			$\{Dgt\}$	$\{Loct\}$	$\{Loc(...)\}$	$\{Dgt, Loct\}$
$\{Loct\}$				Ω	Ω	$\{Loct\}$
$\{Loc(...)\}$					Ω	$\{Loc(...)\}$
$\top$						$\top$

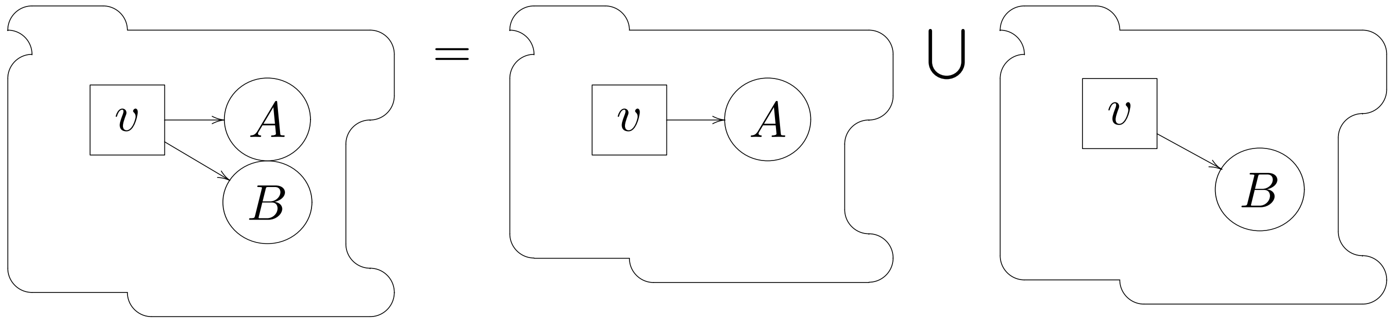
The function sometimes return Ω as the error value.

The initial graph of auxiliary variables



Since $dom(G_0) \cap dom(G_1) = \emptyset$.

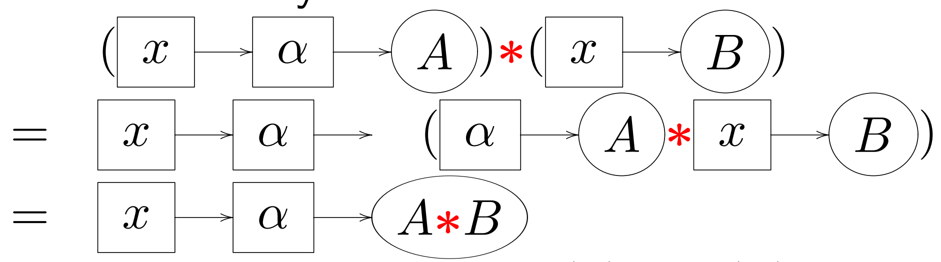
From graph of sets to sets of graph



For v variables (auxiliary or not) not summary nodes.

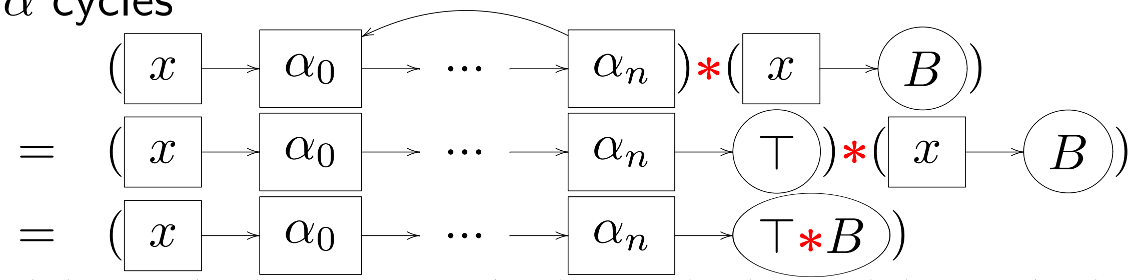
“Jumping” variables (simplified)

- α not summary node



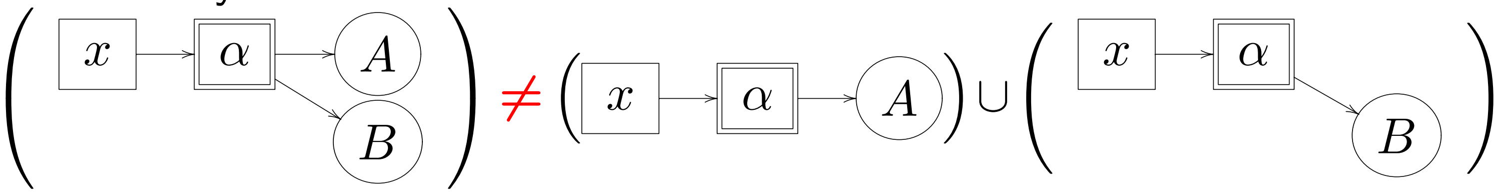
Because $x \rightarrow \alpha$ means $s(x) = f(\alpha)$ so the $*$-information about x can be treated for α instead

- α cycles

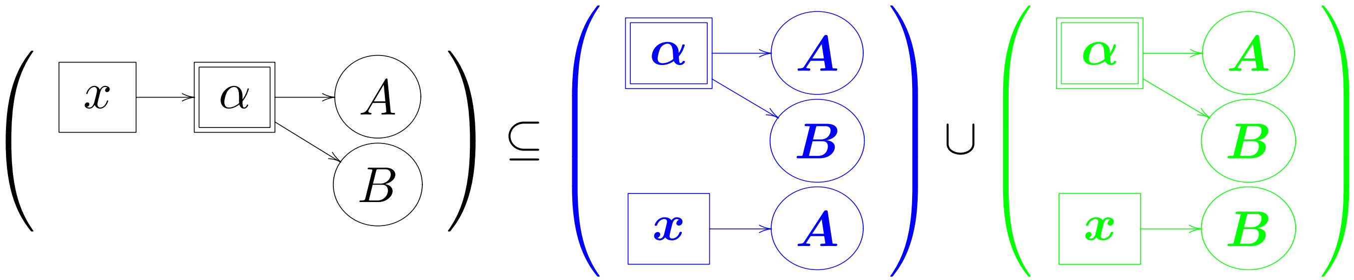


$s(x) = f(\alpha_0) = ... = f(\alpha_n) = f(\alpha_0)$ is $s(x) = f(\alpha_0) = ... = f(\alpha_n)$

- α summary node

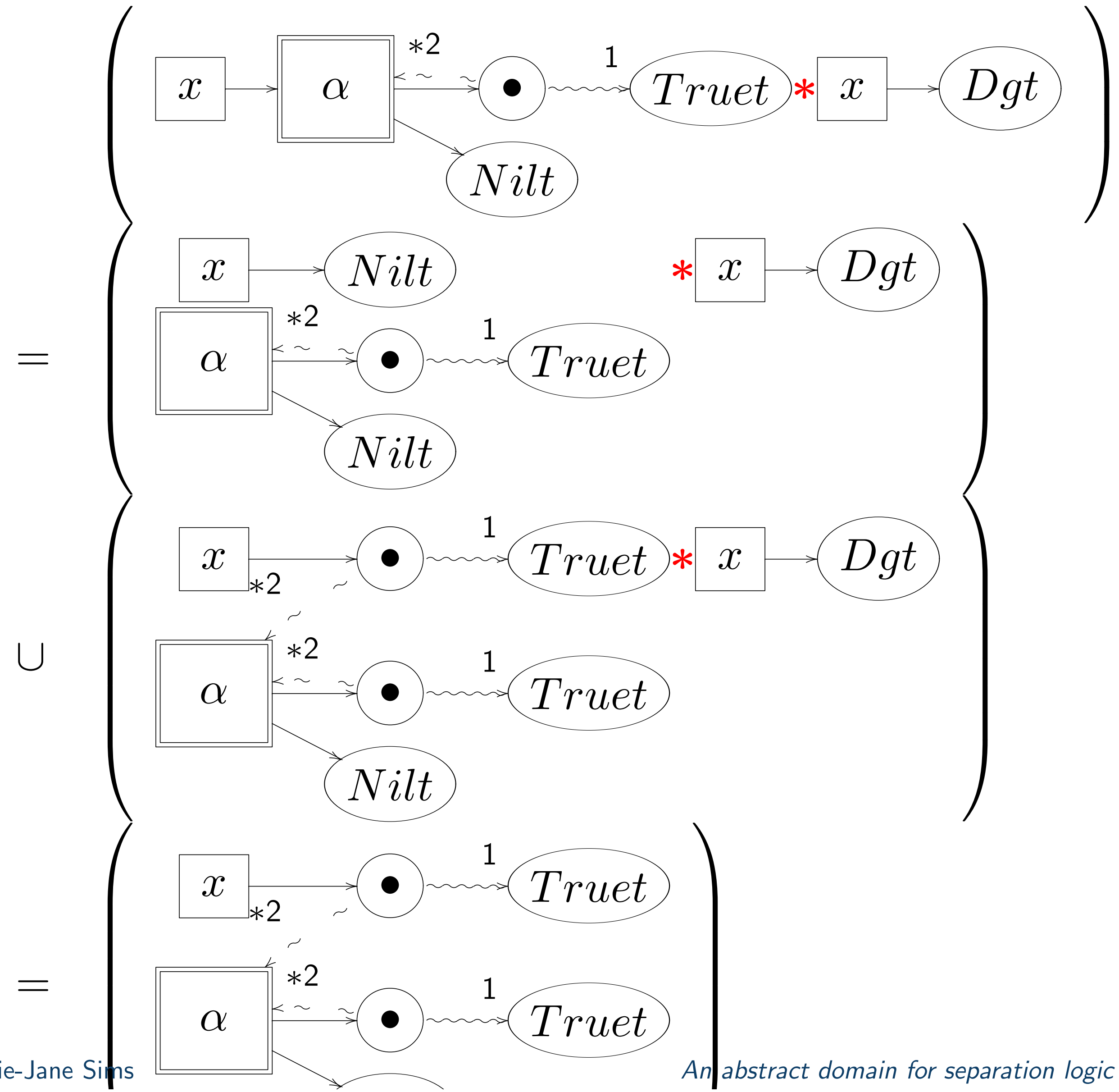


but



Because $s(x) \in f(\alpha) \subseteq (A \cup B) \Rightarrow \boldsymbol{(s(x) \in A \wedge f(\alpha) \subseteq (A \cup B))} \vee \boldsymbol{(s(x) \in B \wedge f(\alpha) \subseteq (A \cup B))}$

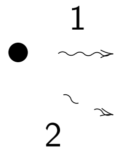
For example, we get



x
α
*2
1
Truet
*
Dgt
Nilt
=
∪

Plan

Comparisons

- the  represent nodes in the usual shape graphs

- summary nodes as for other shape graphs, seems to give more possibilities than predicate abstraction (with each time a specific predicate for list, etc...) but the technics of predicate and their algorithm/heuristics (like folding/unfolding) could probably also be use on our graphs

- a lonely outgoing edge can be seen as a "must" arrow (or valued 1), several outgoing edges from a variable can be seen as a "may" arrow (or valued 1/2, but it is a bit more precise because we know that one of them should exist), and an edge to $\emptyset$ can be seen as a "must not" arrow (or valued 0)

- we deal with numerical (for what we know, only *Magill & al.* also do)
- we have a formal semantic of our domain, the semantics of auxiliary variables are formally defined and formally used in the proofs
- we don't have to check for equalities of variables
- the domain is a cartesian product, we can add or remove some parts depending on the precision we want
- we directly have in the domain the "Dangling" information which is suitable for cleaning checking

End