Conference Record

of the

FOURTH ACM SYMPOSIUM ON
PRINCIPLES OF PROGRAMMING LANGUAGES

Papers Presented at the Symposium
Los Angeles, California
January 17-19, 1977

Sponsored by the
ASSOCIATION FOR COMPUTING MACHINERY
SPECIAL INTEREST GROUP ON AUTOMATA AND COMPUTABILITY THEORY
SPECIAL INTEREST GROUP ON PROGRAMMING LANGUAGES
ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS
OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**
Laboratoire d’Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

1. Introduction

A program denotes computations in some universe of objects. Abstract interpretation of programs consists in using that denotation to describe computations in another universe of abstract objects, so that the results of abstract execution give some informations on the actual computations. An intuitive example (which we borrow from Sintzoff [72]) is the rule of signs. The text \(-1515 \times 17\)
may be understood to denote computations on the abstract universe \((+, -, \#)\) where the semantics of arithmetic operators is defined by the rule of signs. The abstract execution \(-1515 \times 17
\Rightarrow -(+) \times (+) \Rightarrow (-) \times (\#) \Rightarrow (-)\) proves that

Abstract program properties are modeled by a complete semilattice, Birkhoff [61]. Elementary program constructs are locally interpreted by order preserving functions which are used to associate a system of recursive equations with a program. The program global properties are then defined as one of the extreme fixpoints of that system, Tarski [55]. The abstraction process is defined in section 6. It is shown that the program properties obtained by an abstract interpretation of a program are consistent with those obtained by a more refined interpretation of that program. In particular, an abstract interpretation may be shown to be consistent with the formal semantics of the language. Levels of abstraction are formalized by showing that con-
A "computation sequence" with initial state $i_0 \in I$-states is the sequence:
$$s_n = n\text{-state}^n(i_0)$$
for $n = 0, 1, \ldots$
where $f^n$ is the identity function and
$$f^{n+1} = f \circ f^n.$$

The initial to final state transition function:
$$n\text{-state}^\omega : \text{States} \rightarrow \text{States}$$
is the minimal fixpoint of the functional:
$$\lambda F. (n\text{-state} \circ F)$$
Therefore
$$n\text{-state}^\omega = \text{Y} \text{States} \rightarrow \text{States} \left(\lambda F. (n\text{-state} \circ F)\right)$$
where $\text{Y}_F(f)$ denotes the least fixpoint of
$$f : D \rightarrow D$$
Tarski[55].

Since the equation $C_Y(r) = n\text{-context}(r, C_Y)$ must be valid for each arc, $C_Y$ is a solution to the system of "forward" equations:
$$C_Y = F\text{-cont}(C_Y)$$
where
$$F\text{-cont} : \text{Context-Vectors} \rightarrow \text{Context-Vectors}$$
is defined by:
$$F\text{-cont}(C_Y) = \lambda r. n\text{-context}(r, C_Y)$$
Context-Vectors is a complete lattice with union $\cup$ such that $C_Y_1 \cup C_Y_2 = \lambda r. (C_Y_1(r) \cup C_Y_2(r))$.
$F\text{-cont}$ is order preserving for the ordering $\leq$ of Context-Vectors which is defined by:
$$C_Y_1 \leq C_Y_2 \iff \{ \forall r \in \text{Arcs}, C_Y_1(r) \leq C_Y_2(r) \}$$
Hence it is known that $F\text{-cont}$ has fixpoints, Tarski [55]. However, it is trivial to exhibit examples which show that these fixpoints are not always unique. Fortunately, it can be shown that $C_Y$ is
This implies that $A$-$Cont$ is in fact a complete lattice, but we need only one of the two join and meet operations. The set of context vectors is defined by $A$-$Cont = Arcc^0 \rightarrow A$-$Cont$. Whatever $(Cv', Cv'') \in A$-$Cont^2$ may be, we define:

$Cv' \triangleright Cv'' = \lambda r . Cv'(r) \circ Cv''(r)$

$Cv' \lhd Cv'' = \{ \forall r \in Arcc^0, Cv'(r) \leq Cv''(r) \}$

$\triangleright = \lambda r . \top$ and $\lhd = \lambda r . \bot$

$(A$-$Cont, \triangleright, \lhd, \triangleright, \lhd)$ can be shown to be a complete lattice. The function:

$\text{Int} : Arcc^0 \times A$-$Cont \rightarrow A$-$Cont$

defines the interpretation of basic instructions. If $(C(q) | q \in a$-$prod(n))$ is the set of input contexts of node $n$, then the output context on exit arc $r$ of $n$ ($r \in a$-$suc$$(n)$) is equal to $\text{Int}(r, C)$. $\text{Int}$ is supposed to be order-preserving:

For $\forall a \in Arcc, \forall (Cv', Cv'') \in A$-$Cont^2$,

$\{(Cv' \triangleright Cv'') \Rightarrow (\text{Int}(a, Cv') \leq \text{Int}(a, Cv''))\}$

The local interpretation of elementary program constructs which is defined by $\text{Int}$ is used to associate a system of equations with the program. We define

$\text{Int} : A$-$Cont \rightarrow A$-$Cont | \text{Int}(Cv) = \lambda r . \text{Int}(r, Cv)$

It is easy to show that $\text{Int}$ is order-preserving. Hence it has fixpoints, $\text{Tarski}[55]$. Therefore the context vector resulting from the abstract interpretation of the program $P$, which defines the global properties of $P$, may be chosen to be one of the extreme solutions to the system of equations $Cv = \text{Int}(Cv)$.

5.3 Examples

5.3.1 Static Semantics of Programs

The static semantics of programs we defined in section 4 is an abstract interpretation:

$I_{SS} = <\text{Contexts, } \triangleright, \lhd, \text{Env, } \emptyset, \text{ n-context}>$

where Contexts, $\triangleright, \lhd, \text{Env, } \emptyset, \text{ n-context, Context-Vectors, } \triangleright, \lhd, \text{ F-Cont}$ respectively correspond to $A$-$Cont, \triangleright, \lhd, \triangleright, \lhd, \text{ Int, A-Cont, } \triangleright, \lhd, \text{ Int}$.  

5.3.2 Data Flow Analysis

Data flow analysis problems (see references in Ullman[75]) may be formalized as abstract interpretations of programs.
The determination of available expressions, back-dominators, intervals, ... requires a forward system of equations. Some global flow problems, notably the live variables and very busy expressions require propagating information backward through the program graph, they are examples of backward systems of equations.

5.2.3 Remark

Our formal definition of abstract interpretations has the completeness property since the model ensures the existence of a particular solution to the system of equations and therefore defines at least some global property of the program. It must also have the consistency property, that is define only correct properties of programs.

One can distinguish between syntactic and semantic abstract interpretations of a program. Syntactic interpretations are proved to be correct by reference.

Instead of the global hypothesis 6.0 we will use the following local hypothesis on the concrete and abstract interpretations of primitive language constructs:

\[ \begin{align*}
\forall (a, x) \in \text{Arcs} & \times \text{A-Cont}, \\
\gamma(\text{Int}(a, x)) & \succeq \text{Int}(a, \tilde{\gamma}(x))
\end{align*} \]

and

\[ \begin{align*}
\forall (a, x) \in \text{Arcs} & \times \text{C-Cont}, \\
\text{Int}(a, \tilde{\gamma}(x)) & \succeq \alpha(\text{Int}(a, x))
\end{align*} \]

These two hypothesis are in fact equivalent (lemma 1.2 in appendix 12). The following schema illustrates 6.5, i.e. the idea of abstract simulation of concrete computations:

\[ \begin{align*}
\tilde{\gamma} & \quad \text{Int}(a, \tilde{\gamma}) \\
\sim & \quad \text{C}_1 \\
\sim & \quad \gamma
\end{align*} \]
where \( \text{n-pred} \) defines Floyd[67]'s strongest post condition:

\[
\text{n-pred}(r, P) =
\text{let}(n \text{ be origin}(r), (p \text{ be } a\text{-pred(origin}(r)))) \text{ within case } n \text{ in Entries } \\
\text{Junctions } \Rightarrow \text{or } (Pq(n)) \text{ and Tests } \Rightarrow \text{case } r \text{ in } (a\text{-succ}(r(n)) \Rightarrow Pq(p) and test(n) \\
(a\text{-succ}(f(n)) \Rightarrow Pq(p) and not test(n) \\
\text{esac}
\]

Assignments \( \Rightarrow \text{let } (P \text{ be } Pq(p), (x \text{ be id}(n)), (e \text{ be expr}(n))) \text{ within } (\forall v < \text{Values } | P[v/x] \text{ and } x = e[v/x]) \text{ esac} \]

The "invariants" of the program are defined by the least fixpoint of \( \text{n-pred} \) (least for ordering \( \subseteq \)), so that an invariant implies any other correct ass-

The relation \( \equiv \) on abstract interpretations defined by:

\[
\{ I \equiv I' \} \iff \{ (I \leq I') \text{ and } (I' \leq I) \}
\]

is an equivalence relation. We have:

\[
\{ I \equiv (\beta)I' \} \iff \{ \beta \text{ is an isomorphism between the algebras } I \text{ and } I' \}
\]

The proof gives some insight in the abstraction process:

\[
1. \{ I \equiv (\beta)I' \} \iff \{ (I \leq (\beta, \beta^{-1})I') \text{ and } (I' \leq (\beta^{-1}, \beta)I) \}
\]

2. reciprocally,

If \( I \leq (\alpha, \gamma)I' \), let \( \equiv (\alpha, \gamma) \) be the equivalence relation defined on I (properly speaking, on the set of abstract contexts of I) by:

\[
\{ x \equiv (\alpha, \gamma) y \} \iff \{ \alpha(x) = \alpha(y) \}
\]

\( \forall x' \in I' \), each equivalence class \( C_x = \{ x \in I | \alpha_1(x) = x' \} \) has a least upper bound which is \( \gamma_1(x') \). Hence the projection \( \alpha, \gamma_1(I') \) of \( \alpha_1 \) on \( \gamma_1(I') \) is a bijection from the set \( \gamma_1(I') \) of representers of the equivalence classes on I.
A further abstraction may be:
\[
\alpha([a, b]) = \begin{cases} 
    a+b & \text{if } a \equiv b \\
    a & \text{if } a \geq 0 \\
    b & \text{if } b \leq 0 \\
    \text{else if } \text{if } [a, b] = [n, m], \\
    \gamma(\oplus) = [0, +\infty], \\
    \gamma(\ominus) = (-\infty, 0], \\
    \gamma(\otimes) = [-\infty, +\infty].
\end{cases}
\]

The abstract contexts are then:

\[
\Gamma_{CS} = \ldots -4 -3 -2 -1 0 1 2 3 4 \ldots
\]

This interpretation may be abstracted by two non-comparable abstractions:

8. Abstract Evaluation of Programs

The system of equations:
\[
\text{Cv} : \text{Int}(\text{Cv})
\]
resulting from an interpretation \( I = \langle A, \text{Cont}, \equiv \rangle, \leq, \tau, i, \text{Int} \rangle \) of a program \( P \) may be solved by "elimination" methods, (e.g. Tarjan[75]). Otherwise, one can use an "iterative" algorithm which computes Kleene's sequence (L4 of Appendix 12):
\[
\text{Cv} := (C := I; \text{until } C = \text{Int}(C) \text{ do } C := \text{Int}(C) \text{ repeat;} C)
\]

8.1 Correctness

If \( \text{Int} \) is supposed to be a complete morphism (i.e. infinitely distributive over \( \cdot \)) then \( \text{Cv} \) is the least fixpoint of \( \text{Int} \) (e.g. Kildall[75]), since in a semi-lattice of finite length, any distributive function is a complete morphism). Under the weaker assumption that \( \text{Int} \) is continuous, the limit \( \text{Cv} \) of Kleene's sequence can also be shown to be the least fixpoint of \( \text{Int} \) (e.g. Wegbreit[75], since in a well-founded semi-lattice, any isotonum function is continuous). Finally, if \( \text{Int} \) is only supposed to be isotone, \( \text{Cv} \) is an approximation (7) of the least fixpoint (e.g. Kiv and Ullman[75]).

8.2 Termination

The abstract evaluation terminates iff Kleene's
8.4 Example: Performance Analysis of Programs

The performance of programs may be analyzed by deriving for each program point the final value of an imaginary counter which is incremented each time control goes through that point.

Let $A$-Cont be the lattice $\mathbb{R}^+$ of positive real numbers augmented by the upper bound $\infty$, with natural ordering $\leq$. The abstract interpretation:

$$I_P = \langle \mathbb{R}^+, \max, \leq, 0, =, \text{Kir} \rangle$$

may be used to derive the mean values of the counters using Kirchhoff's law of conservation of flow:

$$\text{Kir}(r, CV) = \begin{cases} \frac{p \cdot a \cdot \text{pred}(n)}{\sum_{p \cdot a \cdot \text{pred}(n)}} & \text{if } CV \text{ is defined} \\ \infty & \text{otherwise} \end{cases}$$

The main difficulty is to obtain the probability $\text{Prob(test(n) = true)}$ of taking the true path at a test node $n$. Suppose the values of these probabilities can be determined (from hypothesis on the input data).

For fixed probabilities, the function Kir is clearly continuous (although it is not a complete morphism) since

$$\text{if } CV_0 \sim CV_1 \sim \ldots \sim CV_n \sim \ldots$$

then

$$\max_{i=0}^{\infty} \frac{\sum_{p \cdot a \cdot \text{pred}(n)} CV_i(p)}{\sum_{p \cdot a \cdot \text{pred}(n)}} = \max_{i=0}^{\infty} \left( \frac{\sum_{p \cdot a \cdot \text{pred}(n)} CV_i(p)}{\sum_{p \cdot a \cdot \text{pred}(n)}} \right)$$

and

$$\max_{i=0}^{\infty} (a \cdot q) = \max_{i=0}^{\infty} (a_i \cdot q).$$

The least fixpoint of Kir is the limit of Kleene's sequence (the length of the sequence is in general infinite):

9. Fixpoints Approximation Methods

When the extreme fixpoints of the system of equations established for an abstract interpretation $I$ of a program $P$ cannot be computed in finitely many steps, they can be approximated. A more abstract interpretation $I$ ($1 \leq I$) may be used for that purpose (e.g., Temmamul61). It is often better to make approximations in $I$, for example by "accelerating the convergence" of Kleene's sequences.

9.1 Finite Iterative and Increasing Approximation of the Least Fixpoint Starting from a Lower Bound

Let $I = \langle A$-Cont, $\ast, \leq, 1, \tau, \text{Int} \rangle$ be an interpretation of $P$. When the least fixpoint $CV$ of Int is unreachable, we look for an upper bound $UB$ of $CV$, since according to the correctness requirement 6.0, $CV \leq \gamma(UB)$ and $CV \leq UB$ implies $CV \leq \gamma(UB)$.

9.1.1 Increasing Approximation Sequence

Let $\text{Int} : A$-Cont $\rightarrow A$-Cont be such that:

$$9.1.1.1 \quad (\forall n \geq 0, C = \text{Int}(C) \ast \text{not}(\text{Int}(C) \leq C) = C \ast \text{not}(\text{Int}(C) \leq C))$$

9.1.1.2 Every infinite sequence $I, A$-Int$(I), \ldots, A$-Int$(I), \ldots$ is strictly increasing.

The approximation sequence $S_0, \ldots, S_n, \ldots$ is recursively defined by:

$$9.1.1.3 \quad S_0 = I$$

$$S_{n+1} = \text{if not}(\text{Int}(S_n) \ast S_n) \text{ then } A$-Int$(S_n)$

$$\text{else } S_n$$

We now prove that $\exists m$ finite such that:

$$S_0 \sim S_1 \sim \ldots \sim S_m = S_{m+1} = \ldots$$

Let $n$ be the least natural number (eventually infinite) such that $S_m = S_{m+1}$. If $e \in \mathbb{D}, n$, we know from 9.1.1.3 that $\text{not}(\text{Int}(S_n) \leq S_n)$. Whence by definition $f(S_n) \neq \gamma(UB)$, $f(S_n) \in (0, 1)$. Therefore $

\text{Int}(S_n) \neq S_n$ for all $n$. But $S_n \leq S_{n+1}$ for all $n$, and the limit $S = \lim_{n \to \infty} S_n$ satisfies $S \leq S_m = S_{m+1} = \ldots$.

...
9.1.2 Generalization of Kleene's Ascending Sequence

When A-Cont satisfies the ascending chain condition one can choose $\widetilde{A}$-Int to be Int and therefore the

As before, we define:

9.1.3.5 $\widetilde{A}$-Int $= \lambda \xi. (\lambda q. A$-$\text{int}(q, \xi))$
\[ x := 1 \]

\[ \text{Note: } C_1 \cup C_4 = [1, 101] \leq C_2 = [1, +\infty] \]

* \( C_3 = [1, 100] \)
* \( C_4 = C_3 \cup [1, 1] \)
  \[ = [1, 100] \cup [1, 1] \]
* \( C_4 = [2, 101] \)

stop on that path.

\[ C_5 = C_2 \cap [101, +\infty] \]
  \[ = [1, +\infty] \cap [101, +\infty] \]
* \( C_5 = [101, +\infty] \)

exit, stop.
(However, we will not artificially truncate the decreasing sequence by imposing an arbitrary upper bound on \( n \)).

Let \( \overline{\text{D-int}} : \text{A-Cont} \to \text{A-Cont} \) be such that:

9.3.2.1 \( \forall C \in \text{A-Cont} \)
\( \{ C \geq \overline{\text{D-int}}(C) \} \implies \{ C \geq \overline{\text{D-int}}(C) \} \)

9.3.2.2 \( \forall C \in \text{A-Cont} \), every infinite sequence \( C, \overline{\text{D-int}}(C), \ldots \) is strictly decreasing.

The truncated decreasing sequence \( S'_{0}, \ldots, S'_{n}, \ldots \) is recursively defined by:

9.3.2.3 \( S'_{0} = S_{m} \)
\( S'_{n+1} = \begin{cases} S'_{n} & \text{if } (S'_{n} \neq \text{Int}(S'_{n})) \text{ and } (S'_{n} \neq \overline{\text{D-int}}(S'_{n})) \\ \text{else} & \overline{\text{D-int}}(S'_{n}) \end{cases} \)

Let us now prove that the truncated decreasing sequence is a finite strictly decreasing chain which terms are greater than \( \overline{\text{Int}} \) the least fixpoint of \( \text{Int} \).

Let \( p \) be the least natural number (eventually
\( \lambda \) is fixed) such that \( S'_{p} = S'_{p+1} \) is finally decreasing.

The limit of the descending sequence \( S'_{0} = \tilde{S}, \ldots, \overline{\text{D-int}}(\tilde{S}) = \tilde{S} \) is an upper bound of the greatest fixpoint of \( \text{Int} \).

9.3.4 Narrowing in Truncated Decreasing Sequence

By analogy with 9.1.3 we define now the narrowing operation in order to build a possible construction of \( \overline{\text{D-int}} \) by local modifications to \( \text{Int} \):

9.3.4.1 \( \Delta : \text{A-Cont} \times \text{A-Cont} \to \text{A-Cont} \)

9.3.4.2 \( \forall (C, C') \in \text{A-Cont}^{2}, \)
\( \{ C \geq C' \} \implies \{ C \geq C \wedge C' \geq C' \} \)

9.3.4.3 Every infinite sequence \( s_{0}, \ldots, s_{n}, \ldots \) of the form \( s_{0} = C_{0}, s_{1} = s_{0} \wedge C_{1}, \ldots, s_{n} = s_{n-1} \wedge C_{n}, \ldots \) for arbitrary abstract contexts \( C_{0}, C_{1}, \ldots, C_{n}, \ldots \) is not strictly decreasing.

The approximated interpretation

\( \overline{\text{D-int}} : \text{Arcs}^{3} \times \text{A-Cont} \to \text{A-Cont} \) is defined by:

9.3.4.4 \( \overline{\text{D-int}} = \lambda (q, C_{v}) . \begin{cases} \text{Arcs} & \text{if } q \in \text{Arcs} \text{ then } \text{Arcs} \\ \text{Int}(q) & \text{else } \text{Int}(q, C_{v}) \end{cases} \)
Thus narrowing just discards infinite bounds and makes no improvement on finite bounds, it satisfies the requirements of 9.3.4. According to 9.3.4.4 the system of equations is modified by:

(2) $C_2 = C_2 \triangle (C_1 \cup C_4)$

The descending approximation sequence is:

$C_2 = C_2 \triangle (C_1 \cup C_4)$

$= [1, +\infty) \triangle ([1, 1] \cup [2, 101])$

$= [1, +\infty) \setminus \{1, 101\}$
Any of the AAS, TDS, DAS, TAS methods may yield a fixpoint fp which is not the fixpoint lfp or gfp of interest. None of these methods can improve fp to reach lfp or gfp, therefore a "fixpoint improvement method" is necessary. It is our feeling that such a method could be designed only when considering that A-cont possesses a richer structure (i.e. for particular applications).

Furthermore, in the AAS, TDS, DAS, TAS sequences the term of rank \( n \) is computed only as a function of the term of rank \( n - 1 \), hence these are "sepa-

Acknowledgments

We wish to thank M. Sintzoff for stimulating discussions. We were very lucky to have F. Blanc do the typing for us.

11. References


Scott[71]. The lattice of flow diagrams. Symp. on Semantics of Programming Languages. Springer-Verlag Lecture Notes in Math. (E. Engeler, ed.), Vol. 188.


Sintzoff[72]. Calculating properties of programs by valuations on specific models. Proc. ACM Conf. on Proving Assertions About Programs.

\{\forall (x, y) \in L^2, \{x \leq y \} \Rightarrow \{f(x) \leq f(y)\}\}

\leftrightarrow \{\forall (x, y) \in L^2, \{f(x \cup y) \geq f(x) \cup f(y)\}\}

(H1): Let \( F \) be an order-preserving function from the complete semi-lattice \( <L, \cup, \leq, \top, \bot> \) in itself.

(H2): Let \( \overline{F} \) be an order-preserving function from the complete semi-lattice \( <L, \overline{\cup}, \overline{\leq}, \overline{\top}, \overline{\bot}> \) in itself.

(L1): The fixpoints of \( F \) form a non-empty complete lattice with supremum \( g \), infimum \( \ell \) such that:

\[ g = \bigvee \{x \in L \mid (x \in L) \wedge (x \leq F(x))\} \]

\[ \ell = \bigwedge \{x \in L \mid (x \in L) \wedge (F(x) \leq x)\} \]

(This result is proved in Tarski[55], pp.286-287). Note that the fixpoints of \( \overline{F} \) need not...
(T1): \( H_1, H_1, H_2, H_3 \) imply that the greatest fixpoints \( g \) and \( \overline{g} \) of \( F \) and \( \overline{F} \) are related by:
\[
(\overline{\alpha(g)} \leq \overline{g}) \quad \text{and} \quad (g \leq \gamma(\overline{g}))
\]

Proof:

The existence of \( g \) and \( \overline{g} \) is stated by (L1).

\[
\begin{align*}
\overline{g} & \geq \overline{\alpha(g)} & \text{trivially} \\
\overline{F} & \geq \overline{\alpha(F(g))} & \text{since } \overline{g} = F(g) \\
\overline{F} & \geq \overline{\alpha(\overline{g})} & \text{H3.1, } \leq \text{ isotone, } \leq \text{ transitive} \\
\overline{\alpha(g)} & \geq \alpha(g) & \text{L3} \\
\gamma(\overline{g}) & \geq \gamma(\alpha(g)) & \text{H2.4} \\
\gamma(\overline{g}) & \geq g & \text{H2.6, } \leq \text{ transitive.}
\end{align*}
\]

Q.E.D.

Replacing \( <g, \overline{g}, \overline{\leq}, \overline{\geq}, F, \overline{F}, \alpha, \gamma, H_3.1, H_2.4, H_2.6> \) respectively by \( <\overline{g}, \overline{\alpha}, \overline{\leq}, \overline{\geq}, \overline{F}, \alpha, \gamma, H_3.2, H_2.3, H_2.5> \) in the above proof, we get the "dual" theorem:

(T2): \( H_1, H_1, H_2, H_3 \) imply that the least fixpoints \( \ell \) and \( \overline{\ell} \) of \( F \) and \( \overline{F} \) are related by:
\[
(\overline{\gamma(\ell)} \geq \ell) \quad \text{and} \quad (\ell \geq \gamma(\overline{\ell}))
\]

According to Scott [71] a subset \( X \subseteq L \) is called directed if every finite subset of \( X \) has an upper bound (in the sense of \( \leq \)) belonging to \( X \). (An obvious example of a directed subset is a non-empty ascending chain.) A function \( f : D \rightarrow D \) is called continuous if whenever \( X \subseteq L \) is directed, then \( f(\{x \mid x \in X\}) = \cup \{f(x) \mid x \in X\} \).

(H4): Let \( F \) be a continuous function from the complete semi-lattice \( \langle L, \leq, \tau, 1 \rangle \) in itself.

(H4): Let \( \overline{F} \) be a continuous function from the complete semi-lattice \( \langle L, \overline{\leq}, \overline{\tau}, \overline{1} \rangle \) in itself.

We note \( F^0(x) = x \) and \( F^{n+1}(x) = F(F^n(x)) \).

(L4): (H4)(H4) implies that \( F(\overline{F}) \) has a least fixpoint \( \overline{F}(\ell) \) which is the limit \( \cup_{i=0}^{\infty} F^i(\ell) \) of the Kleene's sequence \( \ell \leq F(\ell) \leq \ldots \leq F^n(\ell) \leq \ldots \)

(The proof is easy to adapt from Kleene [52]'s proof of the first recursion theorem pp. 348-349).