The Journal of Logic and Algebraic Programming 80 (2011) 481–496

Contents lists available at SciVerse ScienceDirect

The Journal of Logic and Algebraic Programming

journal homepage: www.elsevier.com/locate/jlap

Characterizing minimal semantics-preserving slices of predicate-linear, free, liberal program schemas

Sebastian Danicic [b], Robert M. Hierons [c], Michael R. Laurence [a,*]

[a] Department of Computer Science, Regent Court, 211 Portobello, Sheffield S1 4DP, UK
[b] Department of Computing, Goldsmiths College, University of London, London SE14 6NW, UK
[c] Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex UB8 3PH, UK

ARTICLE INFO

Article history:
Received 10 September 2010
Revised 14 April 2011
Accepted 15 April 2011
Available online 21 April 2011

Keywords:
Program schemas
Herbrand domain
Program slicing
Weiser's algorithm
Free and liberal schemas
Linear schemas

ABSTRACT

A program schema defines a class of programs, all of which have identical statement structure, but whose functions and predicates may differ. A schema thus defines an entire class of programs according to how its symbols are interpreted. A *subschema* of a schema is obtained from a schema by deleting some of its statements. We prove that given a schema S which is predicate-linear, free and liberal, such that the true and false parts of every *if* predicate satisfy a simple additional condition, and a slicing criterion defined by the final value of a given variable after execution of any program defined by S, the minimal subschema of S which respects this slicing criterion contains all the function and predicate symbols 'needed' by the variable according to the data dependence and control dependence relations used in program slicing, which is the symbol set given by Weiser's static slicing algorithm. Thus this algorithm gives predicate-minimal slices for classes of programs represented by schemas satisfying our set of conditions. We also give an example to show that the corresponding result with respect to the slicing criterion defined by termination behaviour is incorrect. This complements a result by the authors in which S was required to be function-linear, instead of predicate-linear.

1. Introduction

A schema represents the statement structure of a program by replacing real functions and predicates by symbols representing them. A schema, S, thus defines a whole class of programs which all have the same structure. Each program can be obtained from S via a domain D and an *interpretation* i which defines a function $f^i : D^n \to D$ for each function symbol f of arity n, and a predicate function $p^i : D^m \to \{\mathsf{T}, \mathsf{F}\}$ for each predicate symbol p of arity m. As an example, Fig. 1 gives a schema S, and the program P of Fig. 2 is defined from S by interpreting the function symbols f, g, h and the predicate symbol p as given by P, with D being the set of integers.

The subject of schema theory is connected with that of program transformation and was originally motivated by the wish to compile programs effectively [6]. Schema theory is also relevant to program slicing. Since program slicing algorithms do not normally take into account the meanings of the functions and predicates of a program, a schema encodes all the information about any program which it defines that is available to such algorithms.

A *subschema* of a schema S is defined to be any schema obtained by deleting statements from S. Given a schema S and a variable v, we wish to find a subschema T of S which satisfies the following condition; given any interpretation and any initial state such that the program defined by S terminates, that defined by T also does, and defines the same final value for

* Corresponding author. Tel.: +44 (0) 114 222 1800; fax: +44 (0) 114 222 1810.
E-mail address: mike.rupen@googlemail.com (M.R. Laurence).

1567-8326/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2011.04.009

$$
\begin{array}{l}
u := h();\\
\mathit{if}\ p(w) \quad \mathit{then}\ \ v := f(u);\\
\qquad\qquad \mathit{else}\ \ v := g();
\end{array}
$$

Fig. 1. Schema S.

$$
\begin{array}{l}
u := 1;\\
\mathit{if}\ w > 1 \quad \mathit{then}\ \ v := u + 1;\\
\qquad\qquad \mathit{else}\ \ v := 2;
\end{array}
$$

Fig. 2. Program P.

v. In this case we say that T is a v-slice of S. We are particularly interested in finding *minimal* v-slices of S (with slices of S ordered according to their sets of symbols [1]).

The main theorem of this paper requires that given any path through a schema S, there is an interpretation and an initial state such that the program thus defined follows this path when executed (the freeness condition) and the same term is not generated more than once as it does so (the liberality condition). These conditions were first defined by Paterson [13]. We also require that the same predicate symbol does not occur more than once in S (the predicate-linearity condition), and that if the same function symbol occurs in both the true and false parts of any *if* predicate,[2] then it assigns to distinct variables in each case. We call schemas satisfying all these conditions *special* schemas. We prove that given a schema S which satisfies these conditions and a variable v, the v-slice of S given by Weiser's static slicing algorithm [17] has the unique minimal set of predicate and function symbols of all v-slices of S. Given a schema, Weiser's algorithm computes the subschema containing only those symbols defined by the transitive closure of the control and backward data dependence relations. We also define an ω-slice of a schema in which termination behaviour defines the slicing criterion, and give an example to show that Weiser's algorithm, modified in a natural way with respect to this slicing criterion, need not give a minimal ω-slice. This is in contrast to the situation for function-linear, free, liberal schemas [9].

Our theorem is a strengthening of the result in [4] in which no symbol was allowed to occur more than once in the schema S (that is, S had to be linear, as opposed to just predicate-linear in this paper).

1.1. Organisation of the paper

In the remainder of this section, we explain how the field of program slicing provides motivation for our results, and we also discuss the history of the study of schemas. In Section 2, we give formally our basic schema definitions. In Section 3 we define formally free and liberal schemas, and also give a simple characterisation of schemas that are both free and liberal, which shows that Weiser's algorithm preserves the property of being both free and liberal for slices. In Section 4 we formally define a subschema of a schema. In Section 5 we formally define the data dependence relations $\underset{S}{\rightsquigarrow}$ and $\overset{\text{final}}{\underset{S}{\rightsquigarrow}}$ for a schema S and define Weiser's labelled symbol set for a schema. We also give examples of cases in which the subschema of a schema containing only the symbols in Weiser's set is not the minimal subschema satisfying the required conditions. In Section 6, we define the notion of a p-couple for a predicate p; that is, a pair of interpretations which differ only on one p-predicate term. In Section 7 we introduce formally the class of special schemas to which our results apply. In Section 8, we prove our main theorems. In Section 9, we give an example to show that the subschema of a special schema given by Weiser's algorithm with respect to termination need not be minimal of all subschemas preserving termination behaviour. In Section 10, we discuss our conclusions.

1.2. Relevance of schema theory to program slicing

The field of (static) program slicing is largely concerned with the design of algorithms which given a program and a variable v, eliminate as much code as possible from the program, such that the subprogram consisting of the remaining code, when executed from the same initial state, will still give the same final value for v as the original program, and preserve termination. One algorithm is thus better than another if it constructs a smaller slice.

Most program slicing algorithms are based on the *program dependence graph* (PDG) of a program. This includes Weiser's algorithm [17], which was, however, expressed in different language. (For a fuller discussion of program slicing algorithms see [2,16].) The PDG of a program is a graph whose vertices are the labelled statements of the program and whose directed edges indicate *control* or *data dependence* of one statement upon another.

Data dependence is defined as follows. We say that in a schema S, a function or predicate symbol x is data dependent upon a function symbol f, written $f \underset{S}{\rightsquigarrow} x$, if x references the variable to which f assigns, and there is a path through S passing

[1] A *symbol* in this paper means a function or predicate symbol in a schema.

[2] If the statement $\mathit{if}\ p(v)\ \mathit{then}\ T_1\ \mathit{else}\ T_2$ occurs in a predicate-linear schema S, then we say that T_1 and T_2 are, respectively, the true and false parts of p in S.

$v := g();$

if $p(u)$ *then* $v := g();$

Fig. 3. Deleting the *if* statement gives a v-slice of this schema.

while $q(w)$ *do* {
$w := h_1(w);$
$u := h_2(u);$
if $p(u)$ *then* {
$v := g_1();$
$u := f(u);$
}
}

Fig. 4. Deleting the assignment $u := f(u);$ gives a v-slice of this linear schema, although $u := f(u);$ lies in Weiser's statement set with respect to v.

through f before passing through x without passing through an intermediate assignment to the same variable as f. The relation statement $f \overset{\text{final}}{\underset{S}{\rightsquigarrow}} v$ is defined analogously for a function symbol f and variable v using terminal path-segments. This definition of the relations $\underset{S}{\rightsquigarrow}$, $\overset{\text{final}}{\underset{S}{\rightsquigarrow}}$ is purely syntactic; feasability of any path is not required for it to hold. Thus $h \underset{S}{\rightsquigarrow} f$, $f \overset{\text{final}}{\underset{S}{\rightsquigarrow}} v$ and $g \overset{\text{final}}{\underset{S}{\rightsquigarrow}} v$ hold for the schema of Fig. 1; $g \overset{\text{final}}{\underset{S}{\rightsquigarrow}} v$ means that there is a path through S passing through g, and not subsequently passing through a later assignment to the variable v before reaching the end of the schema.

Slicing algorithms do not take account of the meanings of the functions and predicates occurring in a program, nor do they exploit the knowledge that the same function or predicate occurs in two different places in a program. This reflects the fact that it is undecidable whether the deletion of a particular line of code from a *program* can affect the final value of a given variable after execution [3]. On the other hand a schema likewise encapsulates the data and control dependence relations of the programs that it represents, but whereas it also does not encode the meanings of its function and predicate symbols, it does record any multiple occurrences of these symbols, and this extra information may sometimes lead to a proof of the existence of smaller slices. As an example, it is obvious that the predicate symbol p and the assignment that it controls may be deleted from the schema of Fig. 3 without preventing termination or changing the final value of v (that is, the resulting subschema is a v-slice in our terminology), but most program slicing algorithms will treat the two occurrences of g as if they were two distinct functions, and therefore will not make any deletion.

However, slicing algorithms taking linear schemas as input may yield more information about a program than algorithms that merely use Weiser's algorithm. As an example, in the schema S of Fig. 4, which will be discussed in further sections, it can be seen that the subschema of S obtained by deleting the assignment with symbol f is a v-slice of S, since the removal of this assignment cannot prevent termination (which is determined solely by the value of w when referenced by q), nor can it prevent the path of execution from passing through g_1 at least once, though it may affect the number of times this happens. However, if the assignment with symbol g_1 is replaced by an assignment $v := g_2(v);$ to give a schema T, then the assignment $u := f(u);$ may not similarly be deleted from T, since this deletion may change the value of v after execution. As an example of an interpretation under which this occurs, suppose that h_1, h_2, f and g_2 are all interpreted as the function $v \mapsto v + 1$ in the domain of integers and $q(0), q(1), p(0), p(1)$ and $p(2)$ map to *true*, whereas $q(v)$ and $p(v)$ map to *false* if $v \geq 2$ or $v \geq 3$, respectively. Execution of S from the initial state in which all variables are set to zero results in a final value of 1 for v, whereas if the assignment $u := f(u);$ is deleted, then the execution path will pass through g_2 on both occasions that it enters the body of q giving a final value of 3 for v. However Weiser's algorithm will treat these two cases identically, and will require f to be in a v-slice in both cases. This is because for S and T, Weiser's set with respect to the variable v must contain g_1 or g_2, respectively, since $g_1 \overset{\text{final}}{\underset{S}{\rightsquigarrow}} v$ and $g_2 \overset{\text{final}}{\underset{T}{\rightsquigarrow}} v$, and p controls g_1 or g_2, respectively, hence Weiser's set must contain p, and $f \underset{S}{\rightsquigarrow} h_2 \underset{S}{\rightsquigarrow} p$ (and similarly for T), thus Weiser's set contains f. Danicic [3] gives other examples of cases of linear schemas for which program slicing algorithms will not give minimal correct subschemas. If the linearity assumption is discarded, then non-minimality can be demonstrated even for loop-free schemas, such as the one in Fig. 3, in which p and both occurrences of g lie in the Weiser symbol set defined by v, but the statement containing p can clearly be deleted without changing the final value of v. These examples motivate the mathematical study of schemas, which may lead to the computation of smaller subschemas than conventional program slicing techniques can achieve.

1.3. Different classes of schemas

Many subclasses of schemas have been defined:

Structured schemas, in which *goto* statements are forbidden, and thus loops must be constructed using *while* statements. *All schemas considered in this paper are structured.*

Linear schemas, in which each function and predicate symbol occurs at most once.

Predicate-linear schemas, which we introduce in this paper, in which each predicate symbol occurs at most once, but which may have more than one occurrence of the same function symbol.

Free schemas, where all paths are executable under some interpretation.

Liberal schemas, in which two assignments along any legal path can always be made to assign distinct values to their respective variables.

Near-liberal schemas, which the authors introduced in [5], in which this non-repeating condition applies only to terms not having the form $g()$ for a function symbol g of zero arity.

We now give examples of schemas satisfying these definitions, and first show that the freeness and liberality conditions on schemas are incomparable. To see this, consider the following two examples of linear schemas. The schema

$$\textit{while}\ p(v)\ \textit{do skip}$$

contains no assignments and is therefore liberal, but it is not free, since there is no choice of interpretation and initial state under which the executed path thus defined passes exactly once through the body of p, since the value of v, and hence the boolean value defined at p cannot change during execution. On the other hand the schema

$$\begin{aligned}&\textit{while}\ q(w)\ \textit{do}\ \ \{\\ &\qquad\qquad w := f(w);\\ &\qquad\qquad x := g();\\ &\qquad\qquad \}\end{aligned}$$

is free, since if f defines the function $w \mapsto w + 1$ over the domain of integers, then w never defines a repeated value when referenced by q, and so q can be interpreted so as to define an executed path that passes any desired number of times through q, but it is not liberal, since the variable x is always assigned the same value at occurrences of g along any executed path. The subschema obtained from it by deleting the assignment $x := g()$; (that is, $\textit{while}\ q(w)\ \textit{do}\ w := f(w);$) is both free and liberal, on the other hand.

The schema in Fig. 4 can also be seen to be free, owing to the self-referencing assignments with symbols f, h_1, h_2, which can be interpreted as the function $w \mapsto w + 1$ over the domain of integers, thus ensuring that the variables u, w referenced by p and q, respectively, never repeat in value. It is not liberal however, since it has a path passing more than once through g_1, along which this assignment defines the same value to v on each occasion. More generally, it is easy to see that no schema having a constant assignment in the body of a *while* predicate can be both free and liberal, since if it is free, then there is an executable path passing twice through this assignment, which clearly assigns the same value to its variable on each occasion.

Two schemas are said to be *equivalent* if they have the same termination behaviour, and give the same final value for every variable, given every symbol interpretation and initial state. The authors have shown [10,11] that it is decidable whether linear, free, liberal schemas are equivalent.

Paterson [13] gave a proof that it is decidable whether a schema is both liberal and free (which we give in Section 3); and since he also gave an algorithm transforming a schema S into a schema T such that T is both liberal and free if and only if S is liberal, it is clearly decidable whether a schema is liberal. It is an open problem whether freeness is decidable for the class of linear schemas. However he also proved, using a reduction from the Post Correspondence Problem, that it is not decidable whether an arbitrary schema is free.

1.4. Previous results on the decidability of schema equivalence

Most previous research on schemas has focused on schema equivalence, as defined in Section 1.3. All results on the decidability of equivalence of schemas are either negative or confined to very restrictive classes of schemas. In particular Paterson [13] proved that equivalence is undecidable for the class of all (unstructured) schemas. He proved this by showing that the halting problem for Turing machines (which is, of course, undecidable) is reducible to the equivalence problem for the class of all schemas. Ashcroft and Manna showed [1] that an arbitrary schema can be effectively transformed into an equivalent structured schema, provided that statements such as $\textit{while}\ \neg p(\mathbf{u})\ \textit{do}\ T$ are permitted; hence Paterson's result shows that any class of schemas for which equivalence can be decided must not contain this class of schemas. Thus in order to get positive results on this problem, it is plainly necessary to define the relevant classes of schema with great care.

Positive results on the decidability of equivalence of schemas include the following; in an early result in schema theory, Ianov [8] introduced a restrictive class of schemas, the Ianov schemas, for which equivalence is decidable. This problem was later shown to be NP-complete [7,14].

Paterson [13] proved that equivalence is decidable for a class of schemas called *progressive schemas*, in which every assignment references the variable assigned by the previous assignment along every legal path.

Sabelfeld [15] proved that equivalence is decidable for another class of schemas called *through schemas*. A through schema satisfies two conditions: firstly, that on every path from an accessible predicate p to a predicate q which does not pass through another predicate, and every variable x referenced by p, there is a variable referenced by q which defines a term containing the term defined by x, and secondly, distinct variables referenced by a predicate can be made to define distinct terms under some interpretation.

In view of the evident difficulty of obtaining positive results on this problem, and the importance of program slicing, it seems sensible to concentrate on trying to decide equivalence for classes of schema pairs in which one schema is a subschema of the other, as was done for a class of near-liberal schemas in [5].

2. Basic definitions for schemas

Throughout this paper, $\mathcal{F}$, $\mathcal{P}$, $\mathcal{V}$ and $\mathcal{L}$ denote fixed infinite sets of *function symbols*, *predicate symbols*, *variables* and *labels*, respectively. We assume a function

$$\mathit{arity} : \mathcal{F} \cup \mathcal{P} \to \mathbb{N}.$$

The arity of a symbol x is the number of arguments referenced by x. Note that in the case when the arity of a function symbol g is zero, g may be thought of as a constant.

The set $\mathit{Term}(\mathcal{F}, \mathcal{V})$ of *terms* is defined as follows:

- each variable is a term,
- if $f \in \mathcal{F}$ is of arity n and $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term.

We refer to a tuple $\mathbf{t} = (t_1, \ldots, t_n)$, where each t_i is a term, as a vector term. We call $p(\mathbf{t})$ a predicate term if $p \in \mathcal{P}$ and the number of components of the vector term $\mathbf{t}$ is $\mathit{arity}(p)$.

We also define F-terms and vF-terms recursively for $F \in \mathcal{F}^*$ and $v \in \mathcal{V}$. Any term $f(t_1, \ldots, t_n)$ is an f-term, and the term v is a v-term. If $g \in \mathcal{F}$ and at least one of the terms $t_1, \ldots, t_n$ is an F-term or vF-term, then the term $g(t_1, \ldots, t_n)$ is an Fg-term, or vFg-term, respectively. Thus any FF'-term is also an F'-term.

Definition 1 (*schemas*)**.** We define the set of all *schemas* recursively as follows. *skip* is a schema. An assignment $y := f^{(l)}(\mathbf{x})$; where $y \in \mathcal{V}, f \in \mathcal{F}, l \in \mathcal{L}$ and $\mathbf{x}$ is a vector of $\mathit{arity}(f)$ variables, is a schema. From these all schemas may be 'built up' from the following constructs on schemas.

sequences; $S' = U_1 U_2 \ldots U_r$ is a schema provided that each U_i for $i \in \{1, \ldots, r\}$ is a schema.

if **schemas;** $S'' = \mathit{if}\, p^{(l)}(\mathbf{x})\ \mathit{then}\ \{T_1\}\ \mathit{else}\ \{T_2\}$ is a schema whenever $p \in \mathcal{P}$, $l \in \mathcal{L}$, $\mathbf{x}$ is a vector of $\mathit{arity}(p)$ variables, and T_1, T_2 are schemas. We call the schemas T_1 and T_2 the *true* and *false* parts of $p^{(l)}$.

while **schemas;** $S''' = \mathit{while}\, q^{(l)}(\mathbf{y})\ \mathit{do}\ \{T\}$ is a schema whenever $q \in \mathcal{P}$, $l \in \mathcal{L}$, $\mathbf{y}$ is a vector of $\mathit{arity}(q)$ variables, and T is a schema. We call T the *body* of the *while* predicate $q^{(l)}$ in S'''. If x is a labelled symbol in T, and there is no labelled *while* predicate $p^{(m)}$ in T which also contains x in its body, then we say that $q^{(l)}$ lies *immediately* above x.

Thus a schema is a word in a language over an infinite alphabet. We normally omit the braces { and } if this causes no ambiguity. Also, we may write $\mathit{if}\, p^{(l)}(\mathbf{x})\ \mathit{then}\ \{T_1\}$ instead of $\mathit{if}\, p^{(l)}(\mathbf{x})\ \mathit{then}\ \{T_1\}\ \mathit{else}\ \{T_2\}$ if $T_2 = \mathit{skip}$.

If no symbol (that is, no element of $\mathcal{F} \cup \mathcal{P}$) appears more than once in a schema S, then S is said to be *linear*. If no element of $\mathcal{P}$ appears more than once in a schema S, then S is said to be *predicate-linear*. We define *function-linear* schemas analogously using the set $\mathcal{F}$.

The labels on function and predicate symbols do not affect the semantics of a schema; they are merely included in order to distinguish different occurrences of the same symbol in a schema; *we always assume that distinct occurrences of a symbol in a schema have distinct labels*. We will often omit labels on symbols in contexts where they need not be referred to, as in Fig. 3, or where a symbol only occurs once in a schema. In particular, our main theorems assume predicate-linear schemas, hence we do not label predicate symbols in Section 8.

We define $\mathit{Symbols}(S) = \mathit{Funcs}(S) \cup \mathit{Preds}(S)$, $\mathit{Funcs}(S)$ and $\mathit{Preds}(S)$ to be the sets of symbols, function symbols and predicate symbols occurring in a schema S. Their labelled counterparts are $\mathit{Symbols}^{\mathcal{L}}(S)$, $\mathit{Funcs}^{\mathcal{L}}(S)$ and $\mathit{Preds}^{\mathcal{L}}(S)$. Also $\mathit{ifPreds}^{\mathcal{L}}(S)$ and $\mathit{whilePreds}^{\mathcal{L}}(S)$ are the sets of all labelled *if* predicates and *while* predicates in S. A schema without predicates (that is, a schema which consists of a sequence of assignments and *skip*s) is called *predicate-free*.

If a schema S contains an assignment $y := f^{(l)}(\mathbf{x})$; then we define $y = \mathit{assign}_S(f^{(l)})$ and $\mathbf{x} = \mathbf{refvec}_S(f^{(l)})$. If $p^{(l)} \in \mathit{Preds}^{\mathcal{L}}(S)$ then $\mathbf{refvec}_S(p^{(l)})$ is defined similarly.

Definition 2 (*the* $\searrow_S$ *relation*)**.** Let S be a schema. If $p^{(l)}$ is a labelled predicate in S and x is any (possibly labelled) symbol, we say that $p^{(l)} \searrow_S x$ holds if x lies in the body of $p^{(l)}$ (if $p^{(l)}$ is a *while* predicate in S) or x lies in the true or false part of

$p^{(l)}$ (if $p^{(l)}$ is an *if* predicate). We may strengthen this by writing $p^{(l)} \searrow_S x(Z)$ for $Z \in \{\mathsf{T}, \mathsf{F}\}$ to indicate the additional condition that x lies in the Z-part of $p^{(l)}$ if $p^{(l)} \in \mathit{ifPreds}^{\mathcal{L}}(S)$, or $p^{(l)} \in \mathit{whilePreds}^{\mathcal{L}}(S)$ (if $Z = \mathsf{T}$).

The relation $\searrow_S$ is the transitive closure of the relation 'controls' in program analysis terminology.

2.1. Paths through a schema

The execution of a program defines a possibly infinite sequence of assignments and predicates. Each such sequence will correspond to a *path* through the associated schema. The set $\Pi^{\omega}(S)$ of paths through S is now given.

Definition 3 (*the set $\Pi^{\omega}(S)$ of paths through S, path-segments of S*)**.** If L is any set, then we write L^* for the set of finite words over L and L^{ω} for the set containing both finite and infinite words over L. If σ is a word, or a set of words over an alphabet, then $\mathit{pre}(\sigma)$ is the set of all finite prefixes of (elements of) σ.

For each schema S the alphabet of S, written $\mathit{alphabet}(S)$ is the set

$$\{\underline{y := f^{(l)}(\mathbf{x})} \mid y := f^{(l)}(\mathbf{x}); \text{ is an assignment in } S\}$$
$$\cup$$
$$\{\underline{p^{(l)}, Z} \mid p^{(l)} \in \mathit{Preds}^{\mathcal{L}}(S) \wedge Z \in \{\mathsf{T}, \mathsf{F}\}\}.$$

We define $\mathit{symbol}(\underline{y := f^{(l)}(\mathbf{x})}) = f$ and $\mathit{symbol}(\underline{p^{(l)}, Z}) = p$.

The words in $\Pi(S) \subseteq (\mathit{alphabet}(S))^*$ are formed by concatenation from the words of subschemas of S as follows:

For *skip*,

$$\Pi(\mathit{skip})$$

is the set containing only the empty word.

For assignments,

$$\Pi(y := f^{(l)}(\mathbf{x});) = \{\underline{y := f^{(l)}(\mathbf{x})}\}.$$

For sequences, $\Pi(S_1 S_2 \ldots S_r) = \Pi(S_1) \ldots \Pi(S_r)$.

For *if* schemas, $\Pi(\mathit{if}\, p^{(l)}(\mathbf{x})\, \mathit{then}\, \{T_1\}\, \mathit{else}\, \{T_2\})$ is the set of all concatenations of $\underline{p^{(l)}, \mathsf{T}}$ with a word in $\Pi(T_1)$ and all concatenations of $\underline{p^{(l)}, \mathsf{F}}$ with a word in $\Pi(T_2)$.

For *while* schemas, $\Pi(\mathit{while}\, q^{(l)}(\mathbf{y})\, \mathit{do}\, \{T\}) = (\underline{q^{(l)}, \mathsf{T}}\, \Pi(T))^* \underline{q^{(l)}, \mathsf{F}}$.

We define $\Pi^{\omega}(S) = \{\sigma \in (\mathit{alphabet}(S))^{\omega} \mid \mathit{pre}(\sigma) \subseteq \mathit{pre}(\Pi(S))\}$. Elements of $\Pi^{\omega}(S)$ are called *paths* through S. Any $\mu \in \mathit{alphabet}(S)^*$ is a *path-segment* (in S) if there are words μ', μ'' such that $\mu'\mu\mu'' \in \Pi(S)$. A *terminal* path-segment of S is a path-segment ν such that $\mu\nu \in \Pi(S)$ for some μ.

2.2. Semantics of schemas

The symbols upon which schemas are built are given meaning by defining the notions of a state and of an interpretation. It will be assumed that 'values' are given in a single set D, which will be called the *domain*. We are mainly interested in the case in which $D = \mathit{Term}(\mathcal{F}, \mathcal{V})$ (the Herbrand domain) and the function symbols represent the 'natural' functions with respect to $\mathit{Term}(\mathcal{F}, \mathcal{V})$.

Definition 4 (*states, (Herbrand) interpretations and the natural state e*)**.** Given a domain D, a *state* is either $\perp$ (denoting non-termination) or a function $\mathcal{V} \to D$. The set of all such states will be denoted by $\mathrm{State}(\mathcal{V}, D)$. An interpretation i defines, for each function symbol $f \in \mathcal{F}$ of arity n, a function $f^i : D^n \to D$, and for each predicate symbol $p \in \mathcal{P}$ of arity m, a function $p^i : D^m \to \{\mathsf{T}, \mathsf{F}\}$. The set of all interpretations with domain D will be denoted $\mathit{Int}(\mathcal{F}, \mathcal{P}, D)$.

We call the set $\mathit{Term}(\mathcal{F}, \mathcal{V})$ of terms the *Herbrand domain*, and we say that a function from $\mathcal{V}$ to $\mathit{Term}(\mathcal{F}, \mathcal{V})$ is a Herbrand state. An interpretation i for the Herbrand domain is said to be *Herbrand* if the functions $f^i : \mathit{Term}(\mathcal{F}, \mathcal{V})^n \to \mathit{Term}(\mathcal{F}, \mathcal{V})$ for each $f \in \mathcal{F}$ are defined as

$$f^i(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

for all n-tuples of terms $(t_1, \ldots, t_n)$.

We define the *natural state* $e : \mathcal{V} \to \mathit{Term}(\mathcal{F}, \mathcal{V})$ by $e(v) = v$ for all $v \in \mathcal{V}$.

Note that an interpretation i being Herbrand places no restriction on the mappings $p^i : (Term(\mathcal{F}, \mathcal{V}))^m \to \{\mathsf{T}, \mathsf{F}\}$ defined by i for each $p \in \mathcal{P}$.

Given a schema S and a domain D, an initial state $d \in \text{State}(\mathcal{V}, D)$ with $d \neq \perp$ and an interpretation $i \in Int(\mathcal{F}, \mathcal{P}, D)$ we now define the final state $\mathcal{M}[\![S]\!]^i_d \in \text{State}(\mathcal{V}, D)$ and the associated path $\pi_S(i, d) \in \Pi^\omega(S)$. In order to do this, we need to define the predicate-free schema associated with the prefix of a path by considering the sequence of assignments and *skips* through which it passes.

Definition 5 (*the schema schema*(σ))**.** Given a word $\sigma \in (alphabet(S))^*$ for a schema S, we recursively define the predicate-free schema $schema(\sigma)$ by the following rules; $schema(\lambda) = skip$ if λ is the empty word, $schema(\sigma \underline{v := f(\mathbf{x})}) = schema(\sigma)\, v := f(\mathbf{x})$; and
$schema(\sigma \underline{p^{(l)}, X}) = schema(\sigma)$.

Lemma 6. *Let S be a schema. If $\sigma \in pre(\Pi(S))$, the set $\{m \in alphabet(S) \mid \sigma m \in pre(\Pi(S))\}$ is one of the following; a singleton containing an underlined assignment, a pair $\{\underline{p^{(l)}, \mathsf{T}}, \underline{p^{(l)}, \mathsf{F}}\}$ where $p^{(l)} \in Preds^{(\mathcal{L})}(S)$, or the empty set, and if $\sigma \in \Pi(S)$ then the last case holds.*

Lemma 6, which was proved in [9, Lemma 6], reflects the fact that at any point in the execution of a program, there is never more than one 'next step' which may be taken, and an element of $\Pi(S)$ cannot be a strict prefix of another.

Definition 7 (*semantics of predicate-free schemas*)**.** Given a state $d \neq \perp$, the final state $\mathcal{M}[\![S]\!]^i_d$ and associated path $\pi_S(i, d) \in \Pi^\omega(S)$ of a schema S are defined as follows:

For *skip*,

$$\mathcal{M}[\![skip]\!]^i_d = d$$

and

$\pi_{skip}(i, d)$ is the empty word.

For assignments,

$$\mathcal{M}[\![y := f^{(l)}(\mathbf{x});]\!]^i_d(v) \quad = \quad \begin{cases} d(v) & \text{if } v \neq y, \\ f^i(d(\mathbf{x})) & \text{if } v = y\,, \end{cases}$$

(where the vector term $d(\mathbf{x}) = (d(x_1), \ldots, d(x_n))$ for $\mathbf{x} = (x_1, \ldots, x_n)$)
and

$$\pi_{y := f^{(l)}(\mathbf{x});}(i, d) \quad = \quad y := f^{(l)}(\mathbf{x}),$$

and for sequences $S_1 S_2$ of predicate-free schemas,

$$\mathcal{M}[\![S_1 S_2]\!]^i_d \quad = \quad \mathcal{M}[\![S_2]\!]^i_{\mathcal{M}[\![S_1]\!]^i_d}$$

and

$$\pi_{S_1 S_2}(i, d) \quad = \quad \pi_{S_1}(i, d)\pi_{S_2}(i, \mathcal{M}[\![S_1]\!]^i_d).$$

This uniquely defines $\mathcal{M}[\![S]\!]^i_d$ and $\pi_S(i, d)$ if S is predicate-free.

In order to give the semantics of a general schema S, first the path, $\pi_S(i, d)$, of S with respect to interpretation, i, and initial state d is defined.

Definition 8 (*the path* $\pi_S(i, d)$)**.** Given a schema S, an interpretation i, and a state, $d \neq \perp$, the path $\pi_S(i, d) \in \Pi^\omega(S)$ is defined by the following condition; for all $\sigma \underline{p^{(l)}, X} \in pre(\pi_S(i, d))$, the equality $p^i(\mathcal{M}[\![schema(\sigma)]\!]^i_d(\mathbf{refvec}_S(p^{(l)}))) = X$ holds.

In other words, the path $\pi_S(i, d)$ has the following property; if a predicate expression $p^{(l)}(\mathbf{refvec}_S(p^{(l)}))$ along $\pi_S(i, d)$ is evaluated with respect to the predicate-free schema consisting of the sequence of assignments preceding that predicate in $\pi_S(i, d)$, then the value of the resulting predicate term given by i 'agrees' with the value given in $\pi_S(i, d)$.

By Lemma 6, this defines the path $\pi_S(i, d) \in \Pi^\omega(S)$ uniquely.

Definition 9 (*the semantics of arbitrary schemas*). If $\pi_S(i, d)$ is finite, we define

$$\mathcal{M}[\![S]\!]^i_d = \mathcal{M}[\![schema(\pi_S(i, d))]\!]^i_d$$

(which is already defined, since $schema(\pi_S(i, d))$ is predicate-free) otherwise $\pi_S(i, d)$ is infinite and we define $\mathcal{M}[\![S]\!]^i_d = \bot$. In this last case we may say that $\mathcal{M}[\![S]\!]^i_d$ is not terminating. Also, for schemas S, T and interpretations i and j we write $\mathcal{M}[\![S]\!]^i_d(\omega) = \mathcal{M}[\![T]\!]^j_d(\omega)$ to mean $\mathcal{M}[\![S]\!]^i_d = \bot \iff \mathcal{M}[\![T]\!]^j_d = \bot$. For convenience, if S is predicate-free and $d : \mathcal{V} \to Term(\mathcal{F}, \mathcal{V})$ is a state then we define unambiguously $\mathcal{M}[\![S]\!]_d = \mathcal{M}[\![S]\!]^i_d$; that is, we assume that the interpretation i is Herbrand if d is a Herbrand state; and we will write $\mathcal{M}[\![\mu]\!]_d$ to mean $\mathcal{M}[\![schema(\mu)]\!]_d$ for any $\mu \in alphabet(S)^*$.

Observe that $\mathcal{M}[\![S_1S_2]\!]^i_d = \mathcal{M}[\![S_2]\!]^i_{\mathcal{M}[\![S_1]\!]^i_d}$ and

$$\pi_{S_1S_2}(i, d) = \pi_{S_1}(i, d)\pi_{S_2}(i, \mathcal{M}[\![S_1]\!]^i_d)$$

hold for all schemas (not just predicate-free ones).

Given a schema S, let $\mu \in pre(\Pi(S))$. We say that μ passes through a predicate term $p(\mathbf{t})$ if μ has a prefix μ' ending in $\underline{p^{(l)}, Y}$ for $Y \in \{\mathsf{T}, \mathsf{F}\}$ such that $\mathcal{M}[\![\mu']\!]_e(\mathbf{refvec}_S(p^{(l)})) = \mathbf{t}$ holds. We say that $p(\mathbf{t}) = Y$ is a *consequence* of μ in this case.

3. Free and liberal schemas

Given an initial state and an interpretation, a path through a schema defines a term $f(\mathbf{t})$ or a predicate term $p(\mathbf{t})$ at each symbol that it encounters. For this paper, we wish to consider the class of schemas for which no term or predicate term is defined more than once along any path, given e as the initial state and assuming that all interpretations are Herbrand.

Definition 10 (*free and liberal schemas*). Let S be a schema.

- If for every $\sigma \in pre(\Pi(S))$ there is a Herbrand interpretation i such that $\sigma \in pre(\pi_S(i, e))$, then S is said to be *free*.
- If for every Herbrand interpretation i and any prefix $\mu\ \underline{v := f^{(l)}(\mathbf{a})}\ \nu\ \underline{w := g^{(m)}(\mathbf{b})} \in pre(\pi_S(i, e))$, we have

$$\mathcal{M}[\![\mu\ \underline{v := f^{(l)}(\mathbf{a})}\ \nu\ \underline{w := g^{(m)}(\mathbf{b})}]\!]_e(v) \neq \mathcal{M}[\![\mu\ \underline{v := f^{(l)}(\mathbf{a})}\ \nu\ \underline{w := g^{(m)}(\mathbf{b})}]\!]_e(w),$$

then S is said to be *liberal*. (If $f \neq g$ then of course this condition is trivially satisfied.)

Thus a schema S is said to be free if for every path through S, there is a Herbrand interpretation which follows it with the natural state e as the initial state, and a schema S is said to be liberal if given any path through S passing through two assignments and a Herbrand interpretation which follows it with e as the initial state, the assignments give distinct values to the variables to which they assign. The definitions of freeness and liberality were first given in [13].

Observe that if a schema S is free, then given a Herbrand interpretation i,

$$\mu\ \underline{p^{(l)}, X}\ \mu'\ \underline{p^{(m)}, Y}\ \in pre(\pi_S(i, e))$$

implies that

$$\mathcal{M}[\![\mu]\!]_e(\mathbf{refvec}_S(p^{(l)})) \neq \mathcal{M}[\![\mu\mu']\!]_e(\mathbf{refvec}_S(p^{(m)}))$$

holds, since otherwise there would be no Herbrand interpretation whose path (for initial state e) has the prefix $\mu\ \underline{p^{(l)}, X}\ \mu'\ \underline{p^{(m)}, \neg X}$. Thus a path through a free schema cannot pass more than once (for initial state e) through the same predicate term. Hence if a Herbrand interpretation i maps only finitely many predicate terms to T, and S is a free schema, then the path $\pi_S(i, e)$ terminates. Similarly, if a schema S is free and predicate-linear, and a Herbrand interpretation j maps finitely many *while* predicate terms in S to T, then the path $\pi_S(j, e)$ terminates.

Proposition 11 demonstrates the use of requiring our schemas to be liberal.

Proposition 11. *Let S, T_1, T_2 be predicate-free schemas and assume that each schema ST_i is liberal. Let $v_1, v_2 \in \mathcal{V}$. If $\mathcal{M}[\![ST_1]\!]_e(v_1) = \mathcal{M}[\![ST_2]\!]_e(v_2)$, then $\mathcal{M}[\![T_1]\!]_e(v_1) = \mathcal{M}[\![T_2]\!]_e(v_2)$ holds.*

Proof. Assume $\mathcal{M}[\![ST_1]\!]_e(v_1) = \mathcal{M}[\![ST_2]\!]_e(v_2)$ holds. We will prove $\mathcal{M}[\![T_1]\!]_e(v_1) = \mathcal{M}[\![T_2]\!]_e(v_2)$ by induction on the number of assignments in T_1. We may assume that each schema ST_i contains an assignment to v_i, since if this holds for exactly one value of i, then a contradiction is obtained, and if it is false for both values of i, then the conclusion follows

immediately. Write

$$\mathcal{M}[\![ST_1]\!]_e(v_1) = \mathcal{M}[\![ST_2]\!]_e(v_2) = f(\mathbf{t})$$

and let $v_i := f_i(\mathbf{u}_i)$; be the last assignment to v_i in ST_i for each i. Clearly $f_1 = f_2 = f$.

- Suppose that in the case of ST_1, this last assignment to v_1 occurs in S. Thus this assignment sets the variable v_1 to $f(\mathbf{t})$. Since ST_2 is liberal and $\mathcal{M}[\![ST_2]\!]_e(v_2) = f(\mathbf{t})$ holds, no assignment in T_2 can set a variable to $f(\mathbf{t})$ along ST_2, hence $v_1 := f(\mathbf{u}_1)$; is also the last assignment to $v_1 = v_2$ in ST_2, and so $\mathcal{M}[\![T_1]\!]_e(v_1) = \mathcal{M}[\![T_2]\!]_e(v_2) = v_1 = v_2$ follows, thus proving the desired result.
- Thus we may assume that the last assignment $v_1 := f(\mathbf{u}_1)$; to v_1 in ST_1 occurs in T_1. Similarly, we may assume that the last assignment $v_2 := f(\mathbf{u}_2)$; to v_2 in ST_2 occurs in T_2. Let u_1 and u_2 be the first components of $\mathbf{u}_1$ and $\mathbf{u}_2$, respectively, and write $T_i = T_i' \; v_i := f_i(\mathbf{u}_i); \; T_i''$ for each i. By the inductive hypothesis applied to S and each T_i', the term $\mathcal{M}[\![T_i']\!]_e(u_i)$ is the same for each i; the Proposition then follows from the analogous result for the other components of each $\mathbf{u}_i$. □

Proposition 11 need not hold for non-liberal schemas; for example, if S and T_1 are both $v := g()$; (so ST_1 is not liberal), $T_2 = skip$ and $v_1 = v_2 = v$.

As mentioned in the introduction, it was proved in [13] that it is not decidable whether an (unstructured) schema is free, but it *is* decidable whether it is liberal, or liberal and free. Theorem 12 proves the latter result for structured schemas. It is an open question as to whether freeness of a linear or function-linear schema is decidable.

Theorem 12 (it is decidable whether a schema is liberal and free)**.** *Let S be a schema. Then S is both liberal and free if and only if for every path-segment $\tilde{x}\mu\tilde{y}$ in S with $\tilde{x}, \tilde{y} \in alphabet(S)$, $symbol(\tilde{x}) = symbol(\tilde{y})$ and such that the same labelled symbol does not occur more than once in $\tilde{x}\mu$ or in $\mu\tilde{y}$, then either $\tilde{x}$ and $\tilde{y}$ reference a different vector of variables, or the path-segment $\tilde{x}\mu$ contains an assignment to a variable referenced by $\tilde{y}$.*
In particular, it is decidable whether a schema is both liberal and free.

Proof. [13]. Assume that S is both liberal and free. Then for any path-segment $\tilde{x}\mu\tilde{y}$ satisfying the conditions given, there is a prefix Θ and a Herbrand interpretation i such that $\Theta\tilde{x}\mu\tilde{y} \in pre(\pi_S(i, e))$, and distinct (predicate) terms are defined when $\tilde{x}$ and $\tilde{y}$ are reached, thus proving the necessity of the condition.
To prove sufficiency, first observe that the 'non-repeating' condition on the letters of the path-segment μ may be ignored, since path-segments that begin and end with letters having the same labelled symbol can be removed from within $\tilde{x}\mu$ and $\mu\tilde{y}$ until it is satisfied. Consider the set of prefixes of $\Pi(S)$ of the form $\Theta\tilde{x}\mu\tilde{y}$ with $symbol(\tilde{x}) = symbol(\tilde{y})$ such that $\tilde{x}\mu\tilde{y}$ satisfies the condition given. By induction on the length of such prefixes, it can be shown that every assignment encountered along such a prefix defines a different term (for initial state e), and the result follows immediately from this.
Since there are finitely many path-segments in S satisfying the conditions given for $\tilde{x}\mu\tilde{y}$ and these can be enumerated, the decidability of liberality and freeness for the set of schemas follows easily. □

4. Subschemas and slicing conditions

Definition 13 (*subschemas of a schema*)**.** The set of subschemas of a schema S is the minimal set of schemas which satisfies the following rules:

- *skip* is a subschema of any schema.
- S_1 and S_2 are both subschemas of any schema S_1S_2.
- If S' is a subschema of S, then $S'T$ and TS' are subschemas of ST and TS, respectively.
- if T' is a subschema of T then *while* $p(\mathbf{u})$ *do* T' is a subschema of *while* $p(\mathbf{u})$ *do* T;
- if T' is a subschema of T then the *if* schema *if* $q(\mathbf{u})$ *then* S *else* T' is a subschema of *if* $q(\mathbf{u})$ *then* S *else* T (the true and false parts may be interchanged in this example);
- a subschema of a subschema of S is itself a subschema of S.

Definition 14 (*the semantic u-slice condition for* $u \in \mathcal{V} \cup \{\omega\}$)**.** Let T be a subschema of a schema S. Then given $u \in \mathcal{V}$, we say that T is a u-slice of S if given any domain D, any state $d : \mathcal{V} \to D$ and any $i \in Int(\mathcal{F}, \mathcal{P}, D)$, $\mathcal{M}[\![S]\!]^i_d \neq \bot \Rightarrow (\mathcal{M}[\![T]\!]^i_d \neq \bot \wedge \mathcal{M}[\![S]\!]^i_d(u) = \mathcal{M}[\![T]\!]^i_d(u))$ holds. We also say that T is an ω-slice of S if given any domain D, any state $d : \mathcal{V} \to D$ and any $i \in Int(\mathcal{F}, \mathcal{P}, D)$, $\mathcal{M}[\![S]\!]^i_d \neq \bot \iff \mathcal{M}[\![T]\!]^i_d \neq \bot$ holds.

Thus the u-slice condition is given in terms of every conceivable domain and initial state; however it is well known that the Herbrand domain is the only one that needs to be considered when considering many schema problems. Theorem 15, which is virtually a restatement of [12, Theorem 4–1], ensures that for slicing purposes, we only need to consider Herbrand interpretations and the natural state e.

Theorem 15. *Let χ be a set of schemas, let D be a domain, let d be a function from the set of variables into D and let i be an interpretation using this domain. Then there is a Herbrand interpretation j such that the following hold.*

(1) *For all $S \in \chi$, the path $\pi_S(j, e) = \pi_S(i, d)$.*
(2) *If $S_1, S_2 \in \chi$ and v_1, v_2 are variables and $\rho_k \in pre(\pi_{S_k}(j, e))$ for $k = 1, 2$ and $\mathcal{M}[\![\rho_1]\!]_e(v_1) = \mathcal{M}[\![\rho_2]\!]_e(v_2)$, then also $\mathcal{M}[\![\rho_1]\!]^i_d(v_1) = \mathcal{M}[\![\rho_2]\!]^i_d(v_2)$ holds.*

Throughout the remainder of the paper, all interpretations will be assumed to be Herbrand.

5. The data dependence relations $\underset{S}{\rightsquigarrow}$ and $\overset{\text{final}}{\underset{S}{\rightsquigarrow}}$ and Weiser's labelled symbol set

Definition 16 formalises the $\underset{S}{\rightsquigarrow}$ and $\overset{\text{final}}{\underset{S}{\rightsquigarrow}}$ relations introduced in Section 1.2.

Definition 16 (*the $\underset{S}{\rightsquigarrow}$ and $\overset{final}{\underset{S}{\rightsquigarrow}}$ relations and parameterised path-segments*)**.** Let S be a schema and let σ be a path-segment in S.

We call σ an F-path-segment, or vF-path-segment for $F \in \mathcal{F}^*$ and $v \in \mathcal{V}$ if $\mathcal{M}[\![\sigma]\!]_e(u)$ for some $u \in \mathcal{V}$ is an F-term, or vF-term, respectively. We also call these path-segments an Fu-path-segment or vFu-path-segment, respectively.

We call $\sigma\, \underline{p^{(l)}, Z}$ an Fp-path-segment or $Fp^{(l)}$-path-segment in S if $\mathcal{M}[\![\sigma]\!]_e(u)$ is an F-term for some $u \in \mathcal{V}$ referenced by $p^{(l)}$ in S. We define $vFp^{(l)}$-path-segments analogously.

We sometimes strengthen these definitions by using *labelled* function symbols in the word F to indicate which labelled assignment in S creates the appropriate subterm of $\mathcal{M}[\![\sigma]\!]_e(u)$. We write $f^{(l)} \underset{S}{\rightsquigarrow} g^{(m)}$ if S contains an $f^{(l)}g^{(m)}$-path-segment for $f \in \mathcal{F}$ and $g \in \mathcal{F} \cup \mathcal{P}$, and write $f^{(l)} \overset{\text{final}}{\underset{S}{\rightsquigarrow}} u$ if S contains a terminal path-segment σ such that $\mathcal{M}[\![\sigma]\!]_e(u)$ is an f-term.

The relations $\rightsquigarrow$ and $\overset{\text{final}}{\rightsquigarrow}$ correspond to the data dependence relation in program slicing. We now give examples of these relations. If S is the schema of Fig. 4, the path-segment $\underline{u := f(u)}\ \underline{q(w), \mathsf{T}}\ \underline{w := h_1(w)}\ \underline{u := h_2(u)}$ in S is both an fh_2-path-segment and a ufh_2-path-segment, and the relation $f \underset{S}{\rightsquigarrow} h_2$ holds. Similarly, the path-segment $\underline{q(w), \mathsf{T}}\ \underline{w := h_1(w)}\ \underline{u := h_2(u)}\ \underline{p(u), \mathsf{T}}$ is a uh_2p-path-segment and an h_2p-path-segment, and $h_2 \underset{S}{\rightsquigarrow} p$ holds. Since $\underline{v := g_1()}\ \underline{u := f(u)}\ \underline{q(w), \mathsf{F}}$ is a terminal path-segment in S, $g_1 \overset{\text{final}}{\underset{S}{\rightsquigarrow}} v$ holds.

Definition 17 (*Weiser's labelled symbol set*)**.** Let S be a schema and let $u \in \{\omega\} \cup \mathcal{V}$. Then we define $\mathcal{N}_S(u) \subseteq Funcs^{\mathcal{L}}(S) \cup Preds^{\mathcal{L}}(S)$ to be the minimal set satisfying the following conditions:

(1) If $f^{(l)} \overset{\text{final}}{\underset{S}{\rightsquigarrow}} u \in \mathcal{V}$, then $f^{(l)} \in \mathcal{N}_S(u)$ holds.
(2) If $u = \omega$ then $whilePreds^{\mathcal{L}}(S) \subseteq \mathcal{N}_S(u)$.
(3) If $x \in \mathcal{N}_S(u)$ and $f^{(l)} \underset{S}{\rightsquigarrow} x$, then $f^{(l)} \in \mathcal{N}_S(u)$ holds.
(4) If $x \in \mathcal{N}_S(u)$ and $p^{(l)} \searrow_S x$ then $p^{(l)} \in \mathcal{N}_S(u)$.

The set $\mathcal{N}_S(u)$ (traditionally only defined for the case in which $u \in \mathcal{V}$, and for programs rather than schemas) is fundamental to most slicing algorithms. It contains all symbols which might conceivably affect the final value of u (if u is a variable) or termination (if $u = \omega$). This assertion is formalised in Theorem 18.

Given a schema S and a set $\Sigma \subseteq Symbols^{\mathcal{L}}(S)$ satisfying $(x \in \Sigma \wedge p^{(l)} \searrow_S x) \Rightarrow p^{(l)} \in \Sigma$, there is a subschema T of S such that $Symbols^{\mathcal{L}}(T) = \Sigma$, obtained from S by deleting all elements of $Symbols^{\mathcal{L}}(S) - \Sigma$ from S. This subschema is easily shown to be unique. In particular, for any $u \in \mathcal{V} \cup \{\omega\}$, every schema S has a unique subschema T satisfying $Symbols^{\mathcal{L}}(T) = \mathcal{N}_S(u)$. By Theorem 12, if S is both free and liberal, then so is T.

Theorem 18. *Let S be any schema, let $u \in \mathcal{V} \cup \{\omega\}$ and let T be a subschema of S. If $Symbols^{\mathcal{L}}(T) = \mathcal{N}_S(u)$, then T is a u-slice of S.*

Proof. Proved in [9, Theorem 18]. □

If S is liberal, free, and function-linear, then a subschema T of S is the u-slice of S with the minimal number of labelled symbols if and only if $Symbols^{\mathcal{L}}(T) = \mathcal{N}_S(u)$ holds, as was proved in [9]; but in general this is false. To see this, consider the

$$
\begin{array}{lll}
w := h(); & & \\
\mathit{if}\ p(w) & \mathit{then} & u := g(); \\
 & \mathit{else} & u := g();
\end{array}
$$

Fig. 5. $h \in \mathcal{N}_S(u)$, but deleting the assignment $w := h()$ gives a u-slice of S.

schema S in Fig. 5. It is clearly irrelevant whether $p(w)$ maps to T or F, and hence the assignment $w := h()$ may be deleted to give a u-slice.

Even if a schema is both free and linear, Weiser's algorithm need not give minimal slices. To see this, consider the linear schema S of Fig. 4 which can easily be seen to be free. Owing to the constant g_1-assignment, S is not liberal; any path entering the true part of p more than once would assign the same value, $g_1()$, to v each time. Since S contains the fh_2p-path-segment $u := f(u)\ \underline{q,\ \mathsf{T}}\ w := h_1(w)\ u := h_2(u)\ \underline{p,\ \mathsf{T}}$, and $p \searrow_S g_1$ and $g_1 \overset{\text{final}}{\underset{S}{\rightsquigarrow}} v$ hold, $f \in \mathcal{N}_S(v)$ follows; but the subschema S' of S in which the assignment $u := f(u)$; is deleted is a v-slice of S, since any interpretation j satisfying $\mathcal{M}[\![S']\!]^j_e(v) \neq \mathcal{M}[\![S]\!]^j_e(v)$ would have to define a path $\pi_S(j, e)$ passing through the f-assignment (since otherwise the deletion of f from S would make no difference to $\mathcal{M}[\![S]\!]^j_e(v)$), and so the value of v would be thus fixed at $g_1()$.

6. Couples of interpretations

In order to establish which predicate symbols of a schema must be included in a slice in order to preserve our desired semantics, we define the notion of a p-couple for a predicate p.

Definition 19 (*couples*)**.** Let i, j be interpretations and let $p \in \mathcal{P}$. We say that the set $\{i, j\}$ is a p-couple if there is a vector term $\mathbf{t}$ such that i and j differ only at the predicate term $p(\mathbf{t})$. In this case we may also say that $\{i, j\}$ is a $p(\mathbf{t})$-couple. If a component of $\mathbf{t}$ is an F-term for $F \in \mathcal{F}^*$, then $\{i, j\}$ is an Fp-couple. Given any $u \in \mathcal{V}$ and schema S, we also say that $\{i, j\}$ is an Fpu-couple or $p(\mathbf{t})u$-couple for S if also $\mathcal{M}[\![S]\!]^i_e(u) \neq \mathcal{M}[\![S]\!]^j_e(u)$ and both sides terminate. Lastly, we may label p (an $Fp^{(l)}u$-couple, or $p^{(l)}(\mathbf{t})u$-couple for S) to indicate that the paths $\pi_S(i, e)$ and $\pi_S(j, e)$ diverge at $p^{(l)}$ (at which point the predicate term $p(\mathbf{t})$ is defined).

We also make analogous definitions if instead $u = \omega$; that is, if a set $\{i, j\}$ is a p-couple for a predicate symbol p, then we say $\{i, j\}$ is a $p\omega$-couple for a schema S if exactly one path in $\{\pi_S(i, e), \pi_S(j, e)\}$ terminates.

Note that a pu-couple is simply an Fpu-couple with F as the empty word. The existence of a pu-couple for a schema S 'witnesses' the fact that p affects the semantics of S, as defined by u. As an example of a p-couple, let i be an interpretation that maps the predicate terms $q(w)$, $q(h_1(w))$ and $p(h_2(u))$ to T, and maps $q(h_1(h_1(w)))$ and $p(h_2(f(h_2(u))))$ to F and let the interpretation j be identical except that it maps $p(h_2(f(h_2(u))))$ to T. Then $\{i, j\}$ is a p-couple. If S is the schema in Fig. 4, then both paths $\pi_S(i, e)$, $\pi_S(j, e)$ pass twice through the body of q, with $\pi_S(i, e)$ passing through g_1 only on the first occasion, whereas $\pi_S(j, e)$ passes twice through g_1. Since both interpretations define the same final value for v, $\{i, j\}$ is not a pv-couple for S. However, if T is the schema obtained from S by replacing the assignment $v := g_1()$; by $v := g_2(v)$, then $\{i, j\}$ is a pv-couple for T.

Proposition 20 follows immediately from Definition 19.

Proposition 20. *If $u \in \mathcal{V}$ and T is a u-slice of a schema S, then a pu-couple for S is also a pu-couple for T.*

Definition 21 (*head and tails of a couple*)**.** Let S be a schema. Let $u \in \mathcal{V}$, and let $q \in \mathit{Preds}(S)$. Let $I = \{i, j\}$ be a qu-couple for S and write

$$\pi_S(k, e) = \mu \underline{q^{(l)}, Z_k}\ \rho_k$$

for each $k \in I$ and $\{Z_i, Z_j\} = \{\mathsf{T}, \mathsf{F}\}$; then we define $\mathit{tail}_S(I, k) = \rho_k$ for each $k \in I$, and $\mu = \mathit{head}_S(I)$.

The motivation for Definition 21 is given by Lemma 22, which shows that given a pu-couple for a free liberal schema, a new pu-couple may be obtained from it by replacing its head by any prefix leading to p, while keeping the same tails.

Lemma 22 (*Changing the head of a couple*)**.** *Let S be a free liberal schema and let $p^{(l)} \in \mathit{Preds}^{\mathcal{L}}(S)$ and $u \in \mathcal{V}$. Suppose there is a $p^{(l)}u$-couple I for S and a prefix $\mu\ \underline{p^{(l)}, \mathsf{T}}$ in S, then there is a pu-couple I' for S such that $\mu = \mathit{head}_S(I')$ and $\{\mathit{tail}_S(I, k) \mid k \in I\} = \{\mathit{tail}_S(I', k) \mid k \in I'\}$. In particular, if there is a $p^{(l)}u$-couple I for S and S contains an $Fp^{(l)}$-path-segment for $F \in \mathcal{F}^*$, then there exists an $Fp^{(l)}u$-couple I' for S.*

Proof. Write $I = \{i, j\}$. Since S is free, there exist interpretations i', j' defining paths $\mu\underline{p^{(l)}, Ztail_S(I, i)}$ and $\mu\underline{p^{(l)}, \neg Ztail_S(I, j)}$ for $Z \in \{\mathsf{T}, \mathsf{F}\}$, and by Proposition 11, the final value of u after each path is still distinct. Thus it suffices to prove that i', j' need not differ on any predicate term except the p-predicate term defined after μ. However, if this is false, then $q(\mathbf{t}') = Y$ must be a consequence of one of the paths and $q(\mathbf{t}') = \neg Y$ must be a consequence of the other, for some predicate term $q(\mathbf{t}')$ and $Y \in \{\mathsf{T}, \mathsf{F}\}$. Again, since S is free, $q(\mathbf{t}')$ must occur on the tails of both paths, and by Proposition 11 applied to the variables referenced by the appropriate occurrences of q on each path and the prefixes of the paths preceding these occurrences, the same incompatibility would contradict the existence of the $p^{(l)}u$-couple I. Thus we may define $I' = \{i', j'\}$. □

For the remainder of this paper, we use the following terminology with interpretations. If i is an interpretation, $p(\mathbf{t})$ is a predicate term and $X \in \{\mathsf{T}, \mathsf{F}\}$, then $i(p(\mathbf{t}) = X)$ is the interpretation which maps every predicate term to the same value as i except $p(\mathbf{t})$, which it maps to X.

Lemma 22 need not hold for schemas that are not both free and liberal. To see this, consider the free, linear, non-liberal schema S of Fig. 4.
Let the interpretation i satisfy $q^i(t) = \mathsf{T}$ if and only if the term $t = w$, and $p^i(h_2(u)) = \mathsf{T}$. If the interpretation $j = i(p(h_2(u)) = \mathsf{F})$, then $\{i, j\}$ is an h_2pv-couple for S, since $\mathcal{M}[\![S]\!]^i_e(v) = g_1()$ whereas $\mathcal{M}[\![S]\!]^j_e(v) = v$, but there is no fh_2pv-couple for S, although S contains an fh_2p-path-segment, since any interpretation k such that $\pi_S(k, e)$ passes through the f-assignment must satisfy $\mathcal{M}[\![S]\!]^k_e(v) = g_1()$.

7. Restriction to special schemas

In order to prove our main results, we need to exclude from consideration schemas such as the one in Fig. 5. Therefore we will now only consider schemas such that if the same function symbol occurs in both parts of any if predicate, then the occurrences assign to different variables. The utility of this assumption is demonstrated by Proposition 24.

Definition 23 (*Special schemas*)**.** Let S be a predicate-linear free liberal schema. We say that S is *special* if given any $p \in ifPreds(S)$ and $f \in \mathcal{F}$ such that $p \searrow_S f^{(l)}(\mathsf{T})$ and $p \searrow_S f^{(m)}(\mathsf{F})$ hold, $assign_S(f^{(l)}) \neq assign_S(f^{(m)})$ holds.

Fig. 6 in Section 9 gives an example of a special schema.

Proposition 24. *Let $v \in \mathcal{V}$ and let R, S_1, S_2 be predicate-free schemas such that either S_1 or S_2 contains an assignment to v, each schema RS_j is liberal and for all $f \in \mathcal{F}$, if S_1 and S_2 both contain assignments with function symbol f, then they assign to different variables. Then $\mathcal{M}[\![RS_1]\!]_e(v) \neq \mathcal{M}[\![RS_2]\!]_e(v)$ holds.*

```
x := c();
if p(x) then    {
                u := g1();
                v := g2();
                }
     else       {
                v := g1();
                u := g2();
                }
w := f(u);
while q(w) do   {
                w := f(v);
                a := h(a);
                v := k(a);
                }
```

Fig. 6. Deleting the assignment $x := c()$; gives an ω-slice of this special schema, although $c \in \mathcal{N}_S(\omega)$.

Proof. If only one schema in the set $\{S_1, S_2\}$ contains an assignment to v, then the result follows from the liberality condition. If both do, let f_j be the function symbol of the last assignment to v in each S_j. By our hypotheses, $f_1 \neq f_2$, and each term $\mathcal{M}[\![RS_j]\!]_e(v)$ has f_j as the outermost function symbol, giving the result. □

8. Main theorems

We wish to prove that for any $u \in \mathcal{V}$, every schema which is a u-slice of a given special schema S contains every symbol occurring in $\mathcal{N}_S(u)$. Thus we need to refer to the recursive definition of $\mathcal{N}_S(u)$. This motivates Lemmas 25, 28 and 29, and Definition 26. We first consider Condition (4) in Definition 17, and show that the property of defining a *pu*-couple is 'backward-preserved' by the $\searrow_S$ relation.

Lemma 25. *Let S be a free predicate-linear schema and assume $p \searrow_S q$ for $p, q \in Preds(S)$. Let $u \in \mathcal{V}$. Assume that there exists a qu-couple for S. Then there exists a pu-couple for S.*

Proof. Assume $p \searrow_S q\ (X)$ holds and for each $r \in Preds(S)$, choose $Z_r \in \{\mathsf{T}, \mathsf{F}\}$, subject to the provisos that $Z_p = X$ and $r \in whilePreds(S) \Rightarrow Z_r = \mathsf{T}$. Since $X = \mathsf{T}$ if p is a *while* predicate, this is possible.

Since there exists a *qu*-couple for S, a pair of interpretations (i, j) can be chosen such that $\{i, j\}$ is a *qu*-couple for S and the number of predicate terms $r(\mathbf{s})$ that i maps to Z_r is minimal for all such pairs; clearly this number is finite, since the path $\pi_S(i, e)$ terminates. The path $\pi_S(i, e)$ must pass through q and hence through $\underline{p, X}$, since $p \searrow_S q\ (X)$ holds, and hence there is a predicate term $p(\mathbf{t})$ which i maps to X. Since $q \neq p$, $p^j(\mathbf{t}) = X$ also holds. Define the interpretations i', j' to be identical to i and j, respectively, except that i', j' both map $p(\mathbf{t})$ to $\neg X$. Thus i' maps fewer predicate terms $r(\mathbf{s})$ to Z_r than i does, and hence by the minimality assumption on i, $\{i', j'\}$ is not a *qu*-couple for S. Hence either

$$\mathcal{M}[\![S]\!]^i_e(u) \neq \mathcal{M}[\![S]\!]^{i'}_e(u) \text{ or } \mathcal{M}[\![S]\!]^j_e(u) \neq \mathcal{M}[\![S]\!]^{j'}_e(u)$$

holds.

By the freeness of S and the fact that i' and j' map finitely many predicate terms $r(\mathbf{s})$ for $r \in whilePreds(S)$ to T, the paths $\pi_S(i', e)$ and $\pi_S(j', e)$ are both terminating, and so either $\{i, i'\}$ or $\{j, j'\}$ is a *pu*-couple for S, giving the result. □

It is convenient to make the following definitions, which merely give an alternative way of expressing Weiser's set.

Definition 26 (*(p, X)-links and v-feeding path-segments*)**.** Let S be a predicate-linear schema.
Let $p \in ifPreds(S)$ and $X \in \{\mathsf{T}, \mathsf{F}\}$. A (p, X)-link in S is a path-segment $\underline{p, X}v$ for some path v in the X-part of p in S.
If $p \in whilePreds(S)$, then the path-segment $\underline{p, \mathsf{F}}$ is called a (p, F)-link in S; and a path-segment in $(\underline{p, \mathsf{T}}\Pi(body_S(p)))^*\underline{p, \mathsf{F}}$ which passes at least once through $\Pi(body_S(p))$ is a (p, T)-link.
Let $p, q \in Preds(S)$ and let $v \in \mathcal{V}$. We say that a path-segment μ in S v-feeds p to q if there exists $X \in \{\mathsf{T}, \mathsf{F}\}$ such that $v\mu\underline{q, \mathsf{T}}$ is a path-segment in S for some (p, X)-link v and $\mathcal{M}[\![\mu]\!]_e(w)$ is a vF-term for some $F \in \mathcal{F}^*$ and q references the variable w.

Proposition 27. *Let S_1, S_2, T be predicate-free schemas and let v, w be variables such that $\mathcal{M}[\![S_1]\!]_e(v) \neq \mathcal{M}[\![S_2]\!]_e(v)$ and assume that $\mathcal{M}[\![T]\!]_e(w)$ is a vG-term for some $G \in \mathcal{F}^*$. Then $\mathcal{M}[\![S_1T]\!]_e(w) \neq \mathcal{M}[\![S_2T]\!]_e(w)$ holds.*

Proof. This follows by induction on the total number of assignments and occurrences of *skip* in T. If $T = skip$ then $v = vG = w$ and the result is straightforward. If $T = T'skip$ or $T = T'\,w' := g(\mathbf{u})$; for $w' \neq w$, then $\mathcal{M}[\![S_iT]\!]_e(w) = \mathcal{M}[\![S_iT']\!]_e(w)$ for each i and so the result follows from the inductive hypothesis applied to T'. Thus we may assume that $T = T'\,w := g(w_1, \ldots, w_m)$;. Hence we may write $G = G'g$ such that for some $j \leq m$, $\mathcal{M}[\![T']\!]_e(w_j)$ is a vG'-term. From the inductive hypothesis applied to T', $\mathcal{M}[\![S_1T']\!]_e(w_j) \neq \mathcal{M}[\![S_2T']\!]_e(w_j)$ holds. Since $\mathcal{M}[\![S_iT]\!]_e(w) = g(\mathcal{M}[\![S_iT']\!]_e(w_1), \ldots, \mathcal{M}[\![S_iT']\!]_e(w_m))$ for each i, the result follows. □

We can now prove that the property of defining a *pu*-couple is 'backward-preserved' by the transitive closure of Conditions (3) and (4) of Definition 17.

Lemma 28. *Let S be a special schema. Let $u, v \in \mathcal{V}$ and $p, q \in Preds(S)$. Assume that there exists a qu-couple for S. Suppose that there exists an assignment to v in the body or in one part of p in S and that there exists a path-segment in S v-feeding p to q. Then there exists a pu-couple for S.*

Proof. Given a fixed pair (p, u), we will assume that the conclusion of the Lemma is false, but that the hypotheses are true for some triple (q, v, σ), where σ is a path-segment in S v-feeding p to q, and will show that this leads to a contradiction. We will assume that the triple (q, v, σ) is chosen such that the path-segment σ is of minimal length such that the hypotheses of the Lemma are satisfied.

For some $X \in \{\mathsf{T}, \mathsf{F}\}$, let ρ be a (p, X)-link passing through an assignment to v and let $\mu\rho\sigma \in pre(\Pi(S))$. By Lemma 22, we can choose a qu-couple $I = \{i, j\}$ for S such that $head_S(I) = \mu\rho\sigma$. We may assume that i and j map finitely many *while* predicate terms to T, since the interpretations define terminating paths. Let m be the total number of r-predicate terms which i and j both map to T, where r is the *while* predicate lying immediately above q if $q \in ifPreds(S)$, or q itself if $q \in whilePreds(S)$. If $q \in ifPreds(S)$ and q does not lie in the body of a *while* predicate, then m and r are undefined. We assume that I is chosen such that if defined, m is minimal for the chosen values of q, v and σ.

Let ρ' be any $(p, \neg X)$-link and let Γ be the set of all pairs $(\tilde{q}(\tilde{\mathbf{t}}), Z)$ such that $\tilde{q}(\tilde{\mathbf{t}}) = Z$ is a consequence of the prefix $\mu\rho'\sigma$, but is not a consequence of $\mu\rho\sigma$, and let the interpretations i', j' be obtained by altering i and j, respectively, in accordance with the pairs in Γ; thus, if $(\tilde{q}(\tilde{\mathbf{t}}), Z) \in \Gamma$ then $\tilde{q}^{i'}(\tilde{\mathbf{t}}) = Z$, otherwise $\tilde{q}^{i'}(\tilde{\mathbf{t}}) = \tilde{q}^{i}(\tilde{\mathbf{t}})$, and similarly for j'. Thus the paths $\pi_S(i', e)$ and $\pi_S(j', e)$ both have $\mu\rho'\sigma$ as a prefix. By the freeness of S, the set Γ does not contain any subset of the form $\{(\tilde{q}(\tilde{\mathbf{t}}), Z), (\tilde{q}(\tilde{\mathbf{t}}), \neg Z)\}$ and so i' and j' are well-defined. We write $I' = \{i', j'\}$. We now show that a contradiction is obtained. The proof proceeds in stages:

(1) For any $(\tilde{q}(\tilde{\mathbf{t}}), Z) \in \Gamma$, we now show that there is no $\tilde{q}u$-couple for S. Assume this is false for some $(\tilde{q}(\tilde{\mathbf{t}}), Z)$. By the definition of Γ, $\tilde{q}(\tilde{\mathbf{t}})$ does not occur on μ, and by Lemma 25 and the fact that $p \neq \tilde{q}$ by the falsity of the conclusion of the Lemma, $\tilde{q}(\tilde{\mathbf{t}})$ does not occur on $\mu\rho'$ either, and so $\mu\rho'\sigma$ has a prefix $\mu\rho'\sigma'\underline{\tilde{q}, Z}$ such that $\tilde{q}$ defines $\tilde{q}(\tilde{\mathbf{t}})$ after $\mu\rho'\sigma'$ and since $\tilde{q}(\tilde{\mathbf{t}}) = Z$ is not a consequence of $\mu\rho\sigma$, replacing ρ by ρ' in $\mu\rho\sigma'$ changes the $\tilde{q}$-predicate term defined after $\mu\rho\sigma'$. Hence for some variable v' in the body or in one part of p, σ' v'-feeds p to $\tilde{q}$, contradicting the minimality of σ.

(2) We now show that I' is a qu-couple for S. Suppose this is false. Since I is a qu-couple for S, either $\mathcal{M}[\![S]\!]^{i}_{e}(u) \neq \mathcal{M}[\![S]\!]^{i'}_{e}(u)$ or the analogous assertion holds for j and j'. However, since S is free, changing i or j at finitely many predicate terms still results in an interpretation defining a terminating path through S, and by (1), does not change the final value of u if the predicate terms have the form $\tilde{q}(\tilde{\mathbf{t}})$ for some $(\tilde{q}(\tilde{\mathbf{t}}), Z) \in \Gamma$, thus contradicting the definitions of i' and j' immediately.

(3) Hence I' is a qu-couple for S. Let $\mathbf{t} = \mathcal{M}[\![\mu\rho\sigma]\!]_e(\mathbf{refvec}_S(q))$; thus i and j differ only at $q(\mathbf{t})$. Clearly i' and j' also differ only at $q(\mathbf{t})$ and so their paths diverge at $q(\mathbf{t})$. Since S is free, $q(\mathbf{t}) = Z$ is not a consequence of $\mu\rho\sigma$ for either Z, and so by (1) and the definition of Γ, $q(\mathbf{t})$ does not occur on $\mu\rho'\sigma$ either. Also, $\mathcal{M}[\![\mu\rho\sigma]\!]_e(w) \neq \mathcal{M}[\![\mu\rho'\sigma]\!]_e(w)$ holds for at least one variable w referenced by q, by the assumptions on ρ and σ and Proposition 24 applied to $schema(\mu)$, $schema(\rho)$ and $schema(\rho')$, and Proposition 27 applied to $schema(\mu\rho)$, $schema(\mu\rho')$ and $schema(\sigma)$, and so q does not define $q(\mathbf{t})$ after $\mu\rho'\sigma$. Thus $\pi_S(i', e)$ and $\pi_S(j', e)$ pass at least twice through q after $\mu\rho'\sigma$, and m and r are defined and $head_S(I') = \mu\rho'\sigma\tau$ for some path-segment τ passing at least once through $\underline{r, \mathsf{T}}$.

(4) Thus by Lemma 22, there exists a qu-couple $\tilde{I} = \{\tilde{i}, \tilde{j}\}$ for S which has the same pair of tails as I' and such that $head_S(\tilde{I}) = \mu\rho'\sigma$. We may assume that each r-predicate term which is not a consequence of either path $\pi_S(\tilde{i}, e)$ or $\pi_S(\tilde{j}, e)$ is mapped to F by both interpretations in $\tilde{I}$. We now show that this 'cutting out' of the path-segment τ passing through $\underline{r, \mathsf{T}}$ from $head_S(I')$ contradicts the minimality of m. By (1) and Lemma 25, the elements of I' map the same number of r-predicate terms to T as those in I do. Thus it suffices to prove that the interpretations in $\tilde{I}$ map fewer r-predicate terms to T than those in I'. By the freeness of S and our assumption on $\tilde{I}$, the number of r-predicate terms mapped to T by both interpretations in $\tilde{I}$ is obtained by adding up the number of occurrences of $\underline{r, \mathsf{T}}$ on $head_S(\tilde{I})$ to those on either tail of $\tilde{I}$, and subtracting the number of r-predicate terms mapped to T occurring on both tails of $\tilde{I}$. The analogous assertion holds for I'. Clearly $head_S(\tilde{I})$ has fewer occurrences of $\underline{r, \mathsf{T}}$ than $head_S(I')$ has. Since I' and $\tilde{I}$ have the same tails, it thus remains only to prove that the same number of r-predicate terms mapping to T occur on both $tail_S(I', i')$ and $tail_S(I', j')$ after $head_S(I')$ as after $head_S(\tilde{I})$, and this follows from Proposition 11, since replacing the prefix $head_S(I')$ by $head_S(\tilde{I})$ preserves equalities between predicate terms occurring along $tail_S(I', i')$ and $tail_S(I', j')$. □

We now use Lemma 28 to prove the existence of a pu-couple where membership of the predicate p in $\mathcal{N}_S(u)$ is witnessed by iteration of Conditions (1) and (4) of Definition 17.

Lemma 29. *Let S be a special schema. Let $u, v \in \mathcal{V}$ and $p \in Preds(S)$. Suppose that there exists an assignment to v in the body or in one part of p in S and that there exists a terminal path-segment σ in S such that for some $G \in \mathcal{F}^*$, $\mathcal{M}[\![\sigma]\!]_e(u)$ is a vG-term. Then there exists a pu-couple for S.*

Proof. Let T be the schema *S if* $q(u)$ *then* $u := g_1()$; *else* $u := g_2()$;, where q, g_1, g_2 are distinct symbols not occurring in S. Clearly T is special and the path-segment σ v-feeds p to q in T. The result follows from Lemma 28 applied to T. □

We now use the preceding two Lemmas to show that every symbol of $\mathcal{N}_S(u)$ for a special schema S can affect the semantics of S.

Theorem 30. *Let S be a special schema. Let $u \in \mathcal{V}$.*

(1) *For all* $p \in \mathcal{N}_S(u) \cap \mathcal{P}$ *there exists a* pu*-couple for* S.
(2) *For all* $f^{(l)} \in \mathcal{N}_S(u) \cap \mathcal{F}^{(\mathcal{L})}$, *either there exists an interpretation* i *such that the term* $\mathcal{M}[\![S]\!]^i_e(u)$ *contains the symbol* f, *or there exists* $p \in \mathcal{N}_S(u) \cap \mathcal{P}$ *such that there exists a* $p(\mathbf{t})u$*-couple for* S *for some vector term* $\mathbf{t}$ *containing* f.

Proof. Let Θ be the set of all predicates p in S such that there exists a pu-couple for S and let $P = \mathcal{N}_S(u) \cap \mathcal{P}$.

(1) Observe that from Conditions (1, 3, and 4) of Definition 17, P is the minimal subset of $Preds(S)$ satisfying the following two conditions.
- If $p \in Preds(S)$ and $p \searrow_S f^{(l)}$ for a labelled function symbol $f^{(l)}$ and there exists a terminal $f^{(l)}Fu$-path-segment for some $F \in \mathcal{F}^{(\mathcal{L})^*}$, then $p \in P$ holds.
- If $p \in Preds(S)$ and $p \searrow_S f^{(l)}$ for a labelled function symbol $f^{(l)}$ and $q \in P$ and S contains an $f^{(l)}Fq$-path-segment for some $F \in \mathcal{F}^{(\mathcal{L})^*}$, then $p \in P$.

By Lemmas 29 and 28, respectively, Θ also satisfies both these conditions; hence $P \subseteq \Theta$, as required.
(2) If $f^{(l)} \in \mathcal{N}_S(u) \cap \mathcal{F}^{(\mathcal{L})}$, then from Definition 17, one of the following two possibilities must occur.
- There exists an $f^{(l)}Fu$-path-segment for some $F \in \mathcal{F}^{(\mathcal{L})^*}$, in which case by the freeness of S there exists an interpretation i such that the term $\mathcal{M}[\![S]\!]^i_e(u)$ contains the symbol f, as required.
- The schema S contains an $f^{(l)}Fp$-path-segment for some $F \in \mathcal{F}^{(\mathcal{L})^*}$ and $p \in P \subseteq \Theta$ holds by Part (1) of this Theorem, in which case by Lemma 22, there exists a $p(\mathbf{t})u$-couple for S for some vector term $\mathbf{t}$ one of whose components is an fF-term, proving the result. □

The main theorem of the paper follows.

Theorem 31. *Let* S *be a special schema. Let* $u \in \mathcal{V}$ *and let* T *be a subschema of* S.

(1) *If* $Symbols^{\mathcal{L}}(T) = \mathcal{N}_S(u)$ *then* T *is a* u*-slice of* S.
(2) *If* T *is a* u*-slice of* S, *then* T *contains at least one occurrence of every symbol in* $\mathcal{N}_S(u)$. *In particular, if* $Symbols^{\mathcal{L}}(T) = \mathcal{N}_S(u)$, *then no subschema* T' *of* T *satisfying* $T' \neq T$ *is a* u*-slice of* S *unless there exists* $f \in Funcs(T)$ *such that* T *contains at least two occurrences of* f *and* T' *contains at least one, but not all occurrences of* f *lying in* T.

Proof. Part (1) is a restatement of Theorem 18 for the subclass of special schemas. Part (2) follows immediately from Theorem 30 and Proposition 20, and the definition of a u-slice. □

9. Weiser's algorithm does not give minimal ω-slices for special schemas

Theorems 30 and Part (2) of Theorem 31 do not hold if the variable u is replaced by ω. To see this, consider the special schema S of Fig. 6. By iterating Conditions (2, 3, and 4) of Definition 17, it follows that $\mathcal{N}_S(\omega)$ contains both occurrences of each of f, g_1, g_2 and hence also contains p and c, but we now show that there is no $p\omega$-couple for S. For suppose that $\{i, j\}$ is a $p\omega$-couple for S, and so i and j define paths passing different ways through p. Let $\Omega = \{\pi_S(i, e),\ \pi_S(j, e)\}$. Observe that one path in Ω defines the same predicate term on the second occasion that it passes through q as the other does on the first occasion, and that if $n \geq 3$, the two paths in Ω define the same predicate term on the nth occasion that they pass through q. Thus suppose that one path terminates after passing m times through q. If $m \in \{1, 2\}$, then the other also terminates after passing not more than $3 - m$ times through q. If $m \geq 3$, then so does the other after passing not more than m times through q, giving a contradiction. Thus Part (1) of Theorem 30 is false in this case, and hence it follows easily that the subschema of S obtained by deleting the assignment $x := c();$ is an ω-slice.

10. Conclusions and suggestions for further work

We have shown that for any variable u and a special schema S, the subschema T of S containing the set of predicate symbols and labelled function symbols in the 'Weiser set' $\mathcal{N}_S(u)$, and no others, has the minimal set of predicate and function symbols of any u-slice of S.

This leaves open the possibility that there exists a subschema of T that is a u-slice of S and has fewer, but still non-zero, occurrences of some of the function symbols occurring with labels in $\mathcal{N}_S(u)$. It is not clear whether an example of a special schema exists with this property. Further research should investigate this problem. However if S is not special, this can certainly happen, as the example of the free and predicate-linear but non-liberal schema in Fig. 3 shows.

For $u = \omega$, we have shown that the corresponding result fails, as the special schema is shown in Fig. 6. The existence of this special schema does, however, show the strengthening of our main result compared to that of [4]. Further work will also concentrate on obtaining minimal u-slices for larger classes of schemas. In particular, it would be of interest to be able to effectively characterise minimal slices for a reasonable class of schemas containing those in Figs. 3 and 4, which are near-liberal but not liberal. Danicic et al. [5] gives a related decidability result for schema equivalence. In addition, the main

theorem of the paper can almost certainly be generalised to allow slicing criteria according to which the value of a given variable at a particular point within a program must be preserved by a slice, rather than at the end.

Acknowledgements

This work was supported by a grant from the Engineering and Physical Sciences Research Council, Grant EP/E002919/1.

References

[1] E.A. Ashcroft, Z. Manna, Translating program schemas to while-schemas, SIAM J. Comput. 4 (2) (1975) 125–146.
[2] D.W. Binkley, K.B. Gallagher, Program slicing, in: M. Zelkowitz (Ed.), Advances in Computing, vol. 43, Academic Press, 1996, pp. 1–50.
[3] S. Danicic, Dataflow minimal slicing, Ph.D. Thesis, University of North London, UK, School of Informatics, April 1999.
[4] S. Danicic, C. Fox, M. Harman, R. Hierons, J. Howroyd, M.R. Laurence, Static program slicing algorithms are minimal for free liberal program schemas, Comput. J. 48 (6) (2005) 737–748.
[5] S. Danicic, R.M. Hierons, M.R. Laurence, Decidability of strong equivalence for subschemas of linear, free, near-liberal program schemas, J. Logic Algebr. Programming 80 (2011) 92–112.
[6] S. Greibach, Theory of program structures: schemes, semantics, verification, in: Lecture Notes in Computer Science, vol. 36, Springer-Verlag Inc., New York, NY, USA, 1975
[7] H.B. Hunt, R.L. Constable, S. Sahni, On the computational complexity of program scheme equivalence, SIAM J. Comput. 9 (2) (1980) 396–416.
[8] Y.I. Ianov, The logical schemes of algorithms, in: Problems of Cybernetics, vol. 1, Pergamon Press, New York, 1960, pp. 82–140.
[9] M.R. Laurence, Characterising minimal semantics-preserving slices of function-linear, free, liberal program schemas, J. Logic Algebr. Programming 72 (2) (2005) 157–172.
[10] M.R. Laurence, S. Danicic, M. Harman, R. Hierons, J. Howroyd, Equivalence of conservative, free, linear program schemas is decidable, Theoret. Comput. Sci. 290 (2003) 831–862.
[11] M.R. Laurence, S. Danicic, M. Harman, R. Hierons, J. Howroyd, Equivalence of linear, free, liberal, structured program schemas is decidable in polynomial time, Tech. Rep. ULCS-04-014, University of Liverpool, 2004. Available from: <http://www.csc.liv.ac.uk/research/techreports/>.
[12] Z. Manna, Mathematical Theory of Computation, McGraw-Hill, 1974.
[13] M.S. Paterson, Equivalence problems in a model of computation, Ph.D. Thesis, University of Cambridge, UK, 1967.
[14] J.D. Rutledge, On Ianov's program schemata, J. ACM 11 (1) (1964) 1–9.
[15] V.K. Sabelfeld, An algorithm for deciding functional equivalence in a new class of program schemes, J. Theoret. Comput. Sci. 71 (1990) 265–279.
[16] F. Tip, A survey of program slicing techniques, Tech. Rep. CS-R9438, Centrum voor Wiskunde en Informatica, Amsterdam, 1994.
[17] M. Weiser, Program slices: formal, psychological, and practical investigations of an automatic program abstraction method, Ph.D. Thesis, University of Michigan, Ann Arbor, MI, 1979.