View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Goldsmiths Research Online

Available online at www.sciencedirect.com

ScienceDirect

The Journal of Logic and Algebraic Programming 72 (2007) 157–172

THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING

www.elsevier.com/locate/jlap

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

Michael R. Laurence

Department of Computing, Goldsmiths College, University of London, New Cross, London SE14 6NW, UK

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. As defined in this paper, a *slice* of a schema is obtained from a schema by deleting some of its statements. We prove that given a schema S which is function-linear, free and liberal, and a slicing criterion defined by the final value of a given variable after execution of any program defined by S, the minimal slice of S which respects this slicing criterion contains only the 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 minimal slices for programs representable by function-linear, free, liberal schemas. We also prove a similar result with termination behaviour used as a slicing criterion. This strengthens a recent result, in which S was required to be linear, free and liberal, and termination behaviour as a slicing criterion was not considered.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Program schemas; Program slicing; Decidability; Conservative schemas; Liberal schemas; Free schemas; Linear schemas

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 mapping called an *interpretation* which gives meanings to the function and predicate symbols in S. 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. The subject of schema theory is connected with that of program transformation and was originally motivated by the wish to compile programs effectively. Schema theory is also relevant to program slicing, and this is the motivation for the main results of this paper. Since static 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.

We define a *slice* of a schema S to be any schema obtained by deleting statements from S. Thus a slice of a schema is not required to satisfy any semantic condition; it is defined purely syntactically. Given a schema S and a variable v, we are interested in finding a slice T of S which satisfies the following semantic property: 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 v. In this case we say that T is a v-slice of S. This definition captures for schemas the notion of a static

E-mail address: m.laurence@gold.ac.uk

1567-8326/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2007.02.008

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

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.

end-slice for programs, in which the 'program point' defining the slicing criterion is always at the end of the program, and the slicing criterion must hold for all initial states.

We are specifically interested in finding *minimal* v-slices of schemas (with slices of a schema ordered according to their symbol sets).

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). We also require that the same function symbol does not occur more than once in S (the function-linearity condition). The freeness and liberality conditions were first invented by Paterson [1]. We prove that given a schema S which satisfies these conditions and a variable v, a slice T of S is a minimal v-slice of S if and only if T contains only the set of labelled symbols given by Weiser's static slicing algorithm [2]. This set is syntactically defined using only the data and control dependence relations of S. We also define an ω-slice of a schema S to be a slice of S which terminates if and only if S does for every interpretation and initial state, and prove the corresponding result with respect to termination as a slicing criterion.

Our theorem is a strengthening of the result in [3] 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 function-linear in this paper), and slicing with respect to termination equivalence was not studied.

1.1. Organisation of the paper

In the remainder of Section 1, we justify the study of schemas as an aid to program slicing. We then discuss the various subclasses of schemas, including the classes of free and liberal schemas to which our main theorems apply, and mention the definition of schema equivalence, which is the main focus of schema research.

In Section 2, we give the basic definitions of a schema; this includes more formally stated definitions of concepts mentioned in Section 1. In Section 3, we formally define the classes of free and liberal schemas, and in Section 4 we formally define schema slices and the slicing criteria that we want slices to respect. We also state formally that our slicing criteria may be defined in terms of a very particular domain of values, the Herbrand domain whose elements are the terms constructed using function symbols and variables.

In Section 5, we define the data dependence relation and the labelled symbol set first defined by Weiser, here suitably generalised for the case in which termination behaviour is given as a slicing criterion, and prove that the slice of a schema S containing exactly this set of labelled symbols has the same semantics as S. In Section 6 we define a p-couple for predicate p; that is, a pair of interpretations which differ only at one p-predicate term. In Section 7 we prove our main theorems and in Section 8 we discuss conclusions.

1.2. Relevance of schema theory to static 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 program (slice) 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.

Static program slicing algorithms do not work with individual programs, but rather delete code from a program only if such a deletion is valid for every program having the same 'structure' as the input program. In essence, therefore, they work with schemas. Most program slicing algorithms are based on the *program dependence graph* (PDG) of a program. This includes Weiser's algorithm [2], which was, however, expressed in different language. (For a fuller discussion of program slicing algorithms see [4,5].) 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. We say that in a schema S, a symbol or variable x is data dependent upon a function symbol f, written $f \rightsquigarrow_S x$, if x references the variable to which f assigns, and there is a path through S passing through f before passing through x (or in the case that x is a variable, reaching the end of the program) without passing through an intermediate assignment to the same variable as f. Thus $h \rightsquigarrow_S f$, $f \rightsquigarrow_S v$ and $g \rightsquigarrow_S v$ hold for the schema of Fig. 1. This definition of the relation $\rightsquigarrow$ is purely syntactic; feasibility of any such path is not required for it to hold.

Such 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. 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 (although our main theorem shows that this is not the case for the more restricted class of schemas considered in this paper). As an example, in the schema S of Fig. 3, it is obvious that the predicate symbol p and the g-assignment that it controls may be deleted without changing the final value of v, (that is, $v := g()$; is a v-slice of S 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.

In addition, even linear schemas may yield more information about a program than algorithms based on the PDG of a program. As an example, in the schema S of Fig. 4, (which will be discussed further in Sections 5 and 8) it can be seen that the slice of S obtained by deleting the f-assignment is a v-slice of S, whereas if the g_1-assignment in S is replaced by an assignment $v := g_2(v)$; to give a schema T, then the f-assignment may not similarly be deleted from T, since the resulting schema is not a v-slice; but most program slicing algorithms will treat these two cases similarly, and will require f to be in a v-slice in both cases. The example of Fig. 4 is taken from Danicic [6], which also gives other examples of cases of linear schemas for which program slicing algorithms will not give minimal slices. The existence of such examples motivates the study of schemas, since this may lead to the ability to compute smaller slices than conventional program slicing techniques can achieve.

1.3. Classes of schemas and discussion of schema equivalence

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.

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

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

Conservative schemas, in which every assignment is of the form

$$v := f(v_1, \ldots, v_r); \text{ where } v \in \{v_1, \ldots, v_r\}.$$

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

It can be easily shown that all conservative schemas are liberal.

$$v := g();$$
$$\mathit{if}\ p(u) \quad \mathit{then}\ v := g();$$

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

```
while q(w) do
                {
                w := h1(w);
                u := h2(u);
                if p(u)        then
                                    {
                                    v := g1();
                                    u := f(u);
                                    }
                }
```

Fig. 4. Deleting the assignment $u := f(u)$; gives a v-slice of this schema.

Paterson [1] 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 or function-linear schemas. However he also proved, using a reduction from the Post Correspondence Problem, that it is not decidable whether a schema is free.

Two schemas are said to be *equivalent* if they have the same termination behaviour, and give the same final value for every variable, given any symbol interpretation and initial state. With this definition of equivalence, a slice T of a schema S is equivalent to S if and only if it is both an ω-slice and a v-slice of S for every variable v. It has been shown [7] that it is decidable whether free, liberal and linear schemas are equivalent. In [8, Section 2.4] a discussion of schema equivalence results is given.

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. A *symbol* means an element of $\mathcal{F} \cup \mathcal{P}$ in this paper. 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 inductively for $F \in \mathcal{F}^*$. Any term $f(t_1, \ldots, t_n)$ is an f-term. If at least one of the terms $t_1, \ldots, t_n$ is an F-term and $g \in \mathcal{F}$, then the term $g(t_1, \ldots, t_n)$ is an Fg-term. Thus any FF'-term is also an F'-term.

Function and predicate symbols have labels solely 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*. Labels do not affect the semantics of a schema. 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, when discussing function-linear schemas, we make no distinction between symbols and labelled symbols.

Definition 1 (*schemas*). We define the set of all *schemas* recursively as follows. The empty schema Λ 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 $arity(f)$ variables, is a schema. From these all schemas may be 'built up' from the following constructs on schemas.

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

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

while **schemas;** $S''' =$ *while* $q^{(l)}(\mathbf{y})$ *do* $\{T\}$ is a schema whenever $q \in \mathcal{P}$, $l \in \mathcal{L}$, $\mathbf{y}$ is a vector of $arity(q)$ 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 (in S''').

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

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 $\mathcal{F}$ appears more than once in a schema S, then S is said to be *function-linear*.

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

If a schema S contains an assignment $v := f^{(l)}(\mathbf{x})$; then we define $v = assign_S(f^{(l)})$ and $\mathbf{x} = \mathbf{refvec}_S(f^{(l)})$. If $p^{(l)} \in Preds^{(\mathcal{L})}(S)$ 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 $x \in Symbols^{(\mathcal{L})}(S)$, 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 $p^{(l)} \in ifPreds^{(\mathcal{L})}(S)$, or $p^{(l)} \in whilePreds^{(\mathcal{L})}(S)$ (if $Z = \mathsf{T}$).

The relation $\searrow_S$ is the transitive closure of the relation 'controls' in program slicing 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, 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, and we write Λ to refer to the empty word; recall that Λ is also a particular schema. If σ is a word, or a set of words over an alphabet, then $pre(\sigma)$ is the set of all finite prefixes of (elements of) σ.

For each schema S the alphabet of S, written $alphabet(S)$ is the set

$$\{y := f^{(l)}(\mathbf{x}) \mid y := f^{(l)}(\mathbf{x});\ \text{is an assignment in S}\}$$

$$\bigcup$$

$$\{<p^{(l)} = Z> \mid p^{(l)} \in Preds^{(\mathcal{L})}(S) \wedge Z \in \{\mathsf{T}, \mathsf{F}\}\}.$$

We define $symbol(y := f^{(l)}(\mathbf{x})) = f$ and $symbol(<p^{(l)} = Z>) = p$, and similarly $symbol^{(\mathcal{L})}(y := f^{(l)}(\mathbf{x})) = f^{(l)}$ and $symbol^{(\mathcal{L})}(<p^{(l)} = Z>) = p^{(l)}$.

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

$$\Pi(\Lambda) = \Lambda.$$

For assignments,

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

For sequences, $\Pi(S_1 S_2 \dots S_r) = \Pi(S_1) \dots \Pi(S_r)$.
For *if* schemas,

$$\Pi(\textit{ if } p^{(l)}(\mathbf{x}) \textit{ then } \{T_1\} \textit{ else } \{T_2\}) = <p^{(l)} = \mathsf{T}> \Pi(T_1) \bigcup <p^{(l)} = \mathsf{F}> \Pi(T_2).$$

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

We define $\Pi^\omega(S) = \{\sigma \in (alphabet(S))^\omega | pre(\sigma) \subseteq pre(\Pi(S))\}$. Elements of $\Pi^\omega(S)$ are called *paths* through S. Any $\mu \in alphabet(S)^*$ is a *segment* (in S) if there are words μ_1, μ_2 such that $\mu_1 \mu \mu_2 \in \Pi(S)$. If $\mu_2 = \Lambda$ then μ is a *terminal* segment of S.

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 $D = Term(\mathcal{F}, \mathcal{V})$ (the Herbrand domain) and the function symbols represent the 'natural' functions with respect to $Term(\mathcal{F}, \mathcal{V})$.

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 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 $Int(\mathcal{F}, \mathcal{P}, D)$.

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

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

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

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

Observe that if an interpretation i is Herbrand, this does not restrict the mappings $p^i : (Term(\mathcal{F}, \mathcal{V}))^m \to \{\mathsf{T}, \mathsf{F}\}$ defined by i for each $p \in \mathcal{P}$.

It is well known [9, Section 4–14] that Herbrand interpretations are the only ones that need to be considered when considering many schema properties. This fact is stated more precisely in Theorem 15. In particular, our semantic slicing definitions may be redefined in terms of Herbrand domains.

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 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; $schema(\Lambda) = \Lambda$,
$schema(\sigma <v := f^{(l)}(\mathbf{x})>) = schema(\sigma)\, v := f^{(l)}(\mathbf{x})$; and $schema(\sigma <p^{(m)} = X>) = schema(\sigma)$.

Lemma 6. *Let S be a schema. If $\sigma \in pre(\Pi(S))$, the set $\{m \in alphabet(S) | \sigma m \in pre(\Pi(S))\}$ is one of the following; a singleton containing an assignment, a pair $\{<p^{(l)} = \mathsf{T}>, <p^{(l)} = \mathsf{F}>\}$ 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 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.

Proof of Lemma 6. We prove this by induction on the number of labelled symbols in S. There are four cases to consider.

- If S is an assignment or $S = \Lambda$, then the conclusion of the Lemma is clear.
- Suppose $S =$ *if* $q^{(l)}(\mathbf{x})$ *then* T_1 *else* T_2; if $\sigma = \Lambda$, then the conclusion of the Lemma is clear, otherwise σ begins with $\lessdot q^{(l)} = Z\gtrdot$ and the conclusion of Lemma follows from the induction hypothesis applied to T_1 or T_2.
- Suppose $S = S_1 S_2$ with $\Lambda \notin \{S_1, S_2\}$, then either σ is a strict prefix of $\Pi(S_1)$, in which case the conclusion follows from the induction hypothesis applied to S_1, or $\sigma = \sigma_1\tau$ with $\sigma_1 \in \Pi(S_1)$ and $\tau \in pre(\Pi(S_2))$. By the induction hypothesis applied to S_1, this decomposition of σ is unique, and so the conclusion follows from the induction hypothesis applied to S_2.
- Suppose $S =$ *while* $q^{(l)}(\mathbf{y})$ *do* T, then $\sigma = \lessdot q^{(l)} = \mathsf{T}\gtrdot\ \tau_1 \lessdot q^{(l)} = \mathsf{T}\gtrdot\ \tau_2 \lessdot q^{(l)} = \mathsf{T}\gtrdot \cdots \tau_n\alpha$, where each $\tau_i \in \Pi(T)$ for $i < n$ and $\tau_n \in pre(\Pi(T))$, and either $\alpha = \Lambda$ (which must hold if τ_n is a strict prefix of a path through T) or $\alpha = \lessdot q^{(l)} = Z\gtrdot$. By the induction hypothesis applied to T and the fact that the label l does not occur in T, this decomposition of σ is unique. If $\alpha = \lessdot q^{(l)} = \mathsf{F}\gtrdot$, then the conclusion of the Lemma follows immediately; otherwise σ is a strict prefix of a path through S and the conclusion of the Lemma follows from the induction hypothesis applied to T. □

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 Λ,

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

and

$$\pi_\Lambda i, d = \Lambda.$$

For assignments,

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

(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 $\sigma\ \lessdot p^{(l)} = X\gtrdot \in pre(\pi_S i, d)$, 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 $\mathcal{M}[\![S]\!]^i_d = \perp$. 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 $\mathcal{M}[\![S]\!]^i_d = \perp \iff \mathcal{M}[\![T]\!]^j_d = \perp$. 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.

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

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

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

Given a schema S, let d be a Herbrand state and let $\mu \in pre(\Pi(S))$. We say that μ passes through a predicate term $p(\mathbf{t})$ for initial state d if μ has a prefix μ' ending in $<p^{(l)} = Y>$ for $y \in \{\mathsf{T}, \mathsf{F}\}$ such that $\mathcal{M}[\![schema(\mu')]\!]_d(\mathbf{refvec}_S(p^{(l)})) = \mathbf{t}$ holds. We say that $p(\mathbf{t}) = Y$ is a *consequence* of μ if $d = e$.

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 $\mu\ v := f^{(l)}(\mathbf{a})\ \nu\ w := f^{(m)}(\mathbf{b}) \in pre(\pi_S i, e)$, we have

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

then S is said to be *liberal*.

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 [1].

Observe that if a schema S is free, and

$$\mu\ <p^{(l)} = X>\ \mu'\ <p^{(m)} = Y>\ \in pre(\Pi(S)),$$

then

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

holds, since otherwise there would be no Herbrand interpretation whose path (for initial state e) has the prefix $\mu\ <p^{(l)} = X>\ \mu'\ <p^{(m)} = \neg X>$. 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 finitely many predicate terms to T, then the path $\pi_S i, e$ terminates if S is free.

The schemas in Figs. 3 and 4 are both free but not liberal, since in both cases there is a path which is executable in the Herbrand domain and passes twice through a constant assignment $v := g();$, or $v := g_1();$, respectively. The schema *while* $p(v)$ *do* Λ on the other hand, is liberal but not free, since $<p = \mathsf{T}>\ <p = \mathsf{F}> \in \Pi($*while* $p(v)$ *do* $\Lambda)$ but there is no interpretation i such that $<p = \mathsf{T}>\ <p = \mathsf{F}> = \pi_{\textit{while } p(v) \textit{ do } \Lambda} i, e$ holds.

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. This is proved in [8, Proposition 59]. □

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), $T_2 = \Lambda$ and $v_1 = v_2 = v$.

As mentioned in Section 1, it was proved in [1] 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 segment $\tilde{x}\mu\tilde{y}$ in S such that the following conditions both hold,*

- $\{\tilde{x}, \tilde{y}\} \subseteq alphabet(S) \wedge symbol(\tilde{x}) = symbol(\tilde{y})$, *and*
- *the same labeled symbol does not occur more than once in $\tilde{x}\mu$ or in $\mu\tilde{y}$, either* $\mathbf{refvec}_S(symbol^{(\mathcal{L})}(\tilde{x})) \neq \mathbf{refvec}_S(symbol^{(\mathcal{L})}(\tilde{y}))$, *or the 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 [1]. Assume that S is both liberal and free. Then for any segment $\tilde{x}\mu\tilde{y}$ satisfying the conditions given, there exist a Herbrand interpretation i and $\Theta \in pre(\Pi(S))$ and 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 segment $\tilde{x}\mu\tilde{y}$ may be ignored, since segments that begin and end with letters having the same labeled symbol can be removed from within $\tilde{x}\mu$ and $\mu\tilde{y}$ until the condition is satisfied. Now 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 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. Slices and slicing conditions

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

- if $S = S_1S_2S_3$ then S_1S_3, S_1S_2 and S_2S_3 are slices of S;
- if T' is a slice of T then *while* $p(\mathbf{u})$ *do* T' is a slice of *while* $p(\mathbf{u})$ *do* T;
- if T' is a slice of T then the if schema *if* $q(\mathbf{u})$ *then* S *else* T' is a slice of *if* $q(\mathbf{u})$ *then* S *else* T (the true and false parts may be interchanged in this example);
- a slice of a slice of S is itself a slice of S.

Definition 14 (*the semantic u-slice condition for $u \in \mathcal{V} \cup \{\omega\}$*). Let T be a slice of a schema S.

- Given $v \in \mathcal{V}$, we say that T is a v-slice of S if given any domain D, any state $d : \mathcal{V} \to D$ and any $i \in Int(\mathcal{F}, \mathcal{P}, D)$, $\mathcal{M}[\![S]\!]^i_d \neq \perp \Rightarrow (\mathcal{M}[\![T]\!]^i_d \neq \perp \wedge \mathcal{M}[\![S]\!]^i_d(v) = \mathcal{M}[\![T]\!]^i_d(v))$ 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 $i \in Int(\mathcal{F}, \mathcal{P}, D)$, $\mathcal{M}[\![S]\!]^i_e(\omega) = \mathcal{M}[\![T]\!]^i_e(\omega)$ holds.

Thus the u-slice condition for $u \in \mathcal{V} \cup \{\omega\}$ is given in terms of every conceivable domain and initial state; however Theorem 15, which is virtually a restatement of [9, 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 in $Sch(\mathcal{F}, \mathcal{P}, \mathcal{V})$, let D be a domain, let $d \in D$ and let $i \in Int(\mathcal{F}, \mathcal{P}, D)$. Then there is a Herbrand interpretation j such that for all $S \in \chi$, $\pi_S j, e = \pi_S i, d$ and for any $S, T \in \chi$ and variables v, w, $\mathcal{M}[\![S]\!]^j_e(v) = \mathcal{M}[\![T]\!]^j_e(w) \Rightarrow \mathcal{M}[\![S]\!]^i_d(v) = \mathcal{M}[\![T]\!]^i_d(w)$ holds.*

Corollary 16 shows that $D = Term(\mathcal{F}, \mathcal{V})$ and $d = e$ may be assumed in Definition 14.

Corollary 16. *Let S be a schema and let T be a slice of S.*

(1) *Let $v \in \mathcal{V}$ and suppose that for every Herbrand interpretation j such that $\pi_S j, e$ terminates, $\pi_T j, e$ also does and $\mathcal{M}[\![S]\!]^j_e(v) = \mathcal{M}[\![T]\!]^j_e(v)$ holds. Then T is a v-slice of S.*

(2) *Suppose that for every Herbrand interpretation j, $\mathcal{M}[\![S]\!]^j_e(\omega) = \mathcal{M}[\![T]\!]^j_e(\omega)$ holds. Then T is an ω-slice of S.*

Proof. Both Parts follows immediately from Theorem 15. □

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

5. The data dependence relation $\rightsquigarrow_S$ and Weiser's labelled symbol set

We now give formally the definition of the data dependence relation that was first mentioned in Section 1.2.

Definition 17 (*parameterised segments*). Let S be a schema and let $m \geq 2$. For each $i \in \{1, \ldots, m-1\}$, let x_i be a labelled function symbol and let $x_m \in \mathcal{V} \cup Symbols^{(\mathcal{L})}(S)$. Then we define an $x_1 \ldots x_m$-segment of S to be a segment $\rho = \tilde{x_1}\rho_1\tilde{x_2}\rho_2 \ldots \rho_{m-1}\tilde{x_m}$ in S such that for each $i < m$, $\tilde{x_i} \in alphabet(S)$ with labelled symbol x_i and each segment ρ_i does not pass through an assignment to the variable assigned by x_i, and for each i satisfying $1 < i < m$, x_i references x_{i-1}, and

- if $x_m \in Symbols^{(\mathcal{L})}(S)$ then x_m references $assign_S(x_{m-1})$;
- if $x_m \in \mathcal{V}$ then $x_m = assign_S(x_{m-1})$.

We will sometimes refer to an $f_1{}^{(l_1)} \cdots f_n{}^{(l_n)}$-segment as an $f_1 \cdots f_n$-segment.

Definition 18 (*the $\rightsquigarrow_S$ relation*). Let S be a schema. We write $f^{(l)} \rightsquigarrow_S y$ for a function symbol f and $y \in \mathcal{V} \cup Symbols^{(\mathcal{L})}(S)$ if S contains an $f^{(l)}y$-segment.

As an example, if S is the schema of Fig. 4, then the relations $h_1 \rightsquigarrow_S q, h_1 \rightsquigarrow_S h_1, h_2 \rightsquigarrow_S h_2, h_2 \rightsquigarrow_S p, h_2 \rightsquigarrow_S f, f \rightsquigarrow_S h_2$ (but not $f \rightsquigarrow_S p$), $g_1 \rightsquigarrow_S v, h_1 \rightsquigarrow_S w, h_2 \rightsquigarrow_S u$ and $f \rightsquigarrow_S u$ hold.

Observe that an F-term for $F \in \mathcal{F}^*$ is defined by an element of $pre(\Pi(S))$ ending in an F-segment as it passes through the assignment at the end of the prefix.

Definition 19 (*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.

- If either $x \in \mathcal{N}_S(u)$ or $x = u \in \mathcal{V}$, and S contains an $f^{(l)}Fx$-segment for some $F \in \mathcal{F}^{(\mathcal{L})*}$, then also $f^{(l)} \in \mathcal{N}_S(u)$.
- If $u = \omega$ then $whilePreds^{(\mathcal{L})}(S) \subseteq \mathcal{N}_S(u)$.
- If $x \in \mathcal{N}_S(u)$ and $p^{(l)} \searrow_S x$ then $p^{(l)} \in \mathcal{N}_S(u)$.

Thus $\mathcal{N}_S(u)$ is defined by the transitive closure of the relations $\rightsquigarrow_S$ and $\searrow_S$ and the requirement that $u = \omega \Rightarrow whilePreds^{(\mathcal{L})}(S) \subseteq \mathcal{N}_S(u)$ and $f^{(l)} \rightsquigarrow_S u \in \mathcal{V} \Rightarrow f^{(l)} \in \mathcal{N}_S(u)$. This set (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 is stated formally in Theorem 20.

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 slice 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 slice is easily shown to be unique. In particular, for any $u \in \mathcal{V} \cup \{\omega\}$, every schema S has a unique slice T satisfying $Symbols^{(\mathcal{L})}(T) = \mathcal{N}_S(u)$. Theorem 20 uses Corollary 16 to show that T is a u-slice of S.

Theorem 20. *Let S be any schema, let $u \in \mathcal{V} \cup \{\omega\}$ and let T be a slice of S. If $Symbols^{(\mathcal{L})}(T) = \mathcal{N}_S(u)$, then T is a u-slice of S.*

Proof. Given any $\mu \in (alphabet(S))^\omega$, we define $\Delta(\mu)$ to be the word obtained by deleting every letter in μ whose labelled symbol does not lie in $\mathcal{N}_S(u)$. Thus $\Delta(\Pi^\omega(S)) = \Pi^\omega(T)$ follows from the recursive definition of the schema S and the slice T. Let i be an interpretation. We will show that

$$\Delta(\pi_S i, e) \in pre(\pi_T i, e)$$

holds. Suppose this is false; then since $pre(\pi_T i, e)$ contains the empty word, there is a shortest element $\rho \ {<}p^{(l)} = X{>}$ of the set $pre(\Delta(\pi_S i, e)) - pre(\pi_T i, e)$. Thus there exists $\alpha \ {<}p^{(l)} = X{>} \in pre(\pi_S i, e)$ with $\Delta(\alpha) = \rho$. We now show that

$$\mathcal{M}[\![schema(\rho)]\!]_e(\mathbf{refvec}_S(p^{(l)})) = \mathcal{M}[\![schema(\alpha)]\!]_e(\mathbf{refvec}_S(p^{(l)}))$$

holds. Clearly $p^{(l)} \in \mathcal{N}_S(u)$, so this follows from the closure property of $\mathcal{N}_S(u)$ under 'backward data dependence', since the letters deleted from α to obtain ρ do not have labelled symbols in $\mathcal{N}_S(u)$ and therefore do not change the predicate term defined by $p^{(l)}$.

Clearly $\rho \ {<}p^{(l)} = X{>} \in pre(\Delta(\Pi(S)) = pre(\Pi(T))$, and so $\rho \ {<}p^{(l)} = Z{>} \in pre(\pi_T i, e)$ for some $Z \in \{\mathsf{T}, \mathsf{F}\}$ follows from $\rho \in pre(\pi_T i, e)$ using Lemma 6. Since the interpretation i maps the predicate term $p^{\mathcal{M}[\![schema(\alpha)]\!]_e(\mathbf{refvec}_S(\vec{x}))}$ to X, $Z = X$ follows, contradicting the choice of ρ.

Hence we have shown $\Delta(\pi_S i, e) \in pre(\pi_T i, e)$, and since $\Delta(\pi_S i, e) \in \Delta(\Pi^\omega(S)) = \Pi^\omega(T)$, by Lemma 6,

$$\Delta(\pi_S i, e) = \pi_T i, e$$

follows. Thus if $\pi_S i, e$ terminates, so does $\pi_T i, e$.

If $u = \omega$ then the converse is true, since if $\pi_S i, e$ is infinite, it must pass through infinitely many occurrences of a letter ${<}q^{(m)} = \mathsf{T}{>}$ for a while predicate $q^{(m)}$, and Δ does not delete these letters. Thus by Part (2) of Corollary 16, T is an ω-slice of S. If on the other hand $u \in \mathcal{V}$, then $\mathcal{M}[\![schema(\pi_S i, e)]\!]_e(u) = \mathcal{M}[\![schema(\pi_T i, e)]\!]_e(u)$ follows again from the properties of $\mathcal{N}_S(u)$ and Δ, proving by Part (1) of Corollary 16 that T is a u-slice of S. □

We will henceforth refer to a schema which is function-linear, free and liberal as an f-LFL schema.

If S is f-LFL, then a slice T of S is the u-slice of S with the minimal number of labelled symbols if and only if $Symbols^{(\mathcal{L})}{}_S(T) = \mathcal{N}_S(u)$ holds, as we shall prove; but in general this is false. 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-segment $u := f(u) \ {<}q = \mathsf{T}{>} \ w := h_1(w)\, u := h_2(u) \ {<}p = \mathsf{T}{>}$, and $p \searrow_S g$ and $g_1 \rightsquigarrow_S v$ hold, $f \in \mathcal{N}_S(v)$ follows; but the slice 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 $g_1()$. We will show in Theorem 29 that a situation of this kind cannot occur with f-LFL schemas.

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 21 (*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 $u \in \mathcal{V} \cup \{\omega\}$ and schema S, we also say that $\{i, j\}$ is an Fpu-couple or $p(\mathbf{t})u$-couple for S if also $\mathcal{M}[\![S]\!]^i_e(u) \neq \mathcal{M}[\![S]\!]^j_e(u)$ (recall that if $u = \omega$, this means that exactly one side terminates) and if $u \in \mathcal{V}$ then *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).

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.

Proposition 22 follows immediately from Definition 21.

Proposition 22. *If $u \in \mathcal{V} \cup \{\omega\}$ and T is a u-slice of a schema S, then a pu-couple for S is also a pu-couple for T.*

Definition 23 (*head and tails of a couple*). Let S be a schema. Let $u \in \mathcal{V} \cup \{\omega\}$, and let $q \in Preds(S)$. Let $I = \{i, j\}$ be a qu-couple for S and write

$$\pi_S k, e = \mu \lessdot q^{(l)} = Z(k) \gtrdot \rho(k)$$

for each $k \in I$ and $\{Z(i), Z(j)\} = \{\mathsf{T}, \mathsf{F}\}$; then we define $tail_S(k, I) = \rho(k)$ for each $k \in I$, and $\mu = head_S(I)$.

The motivation for Definition 23 is given by Lemma 24, which shows that given a $p^{(l)}u$-couple for a free liberal schema, a new $p^{(l)}u$-couple may be obtained from it by replacing its head by any prefix leading to $p^{(l)}$, while keeping the same tails.

Lemma 24 (Changing the head of a couple). *Let S be a free liberal schema and let $p^{(l)} \in Preds^{(\mathcal{L})}(S)$ and $u \in \mathcal{V} \cup \{\omega\}$. Suppose there is a $p^{(l)}u$-couple I for S and a prefix $\mu \lessdot p^{(l)} = \mathsf{T} \gtrdot$ in S, then there is a $p^{(l)}u$-couple I' for S such that $\mu = head_S(I')$ and $\{tail_S(k, I) | k \in I\} = \{tail_S(k, I') | k \in I'\}$. In particular, if there is a $p^{(l)}u$-couple I for S and S contains an $Fp^{(l)}$-segment for $F \in \mathcal{F}^{(\mathcal{L})*}$, then there exists an $Fp^{(l)}u$-couple I' for S.*

Proof. This is proved in [7, Lemma 22], by applying Proposition 11 to prefixes of $\pi_S i, e$ for each $i \in I$ which end in the same predicate term. □

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 24 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 $q(t)^i = \mathsf{T}$ if and only if the term $t = w$, and $p^i(h_2(u)) = \mathsf{T}$. If the interpretation $j = i(h_2(w) = \mathsf{F})$, 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-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. Main theorems

We wish to show that every schema which is a u-slice of a given f-LFL schema S contains every labelled symbol occurring in $\mathcal{N}_S(u)$. Thus we need to refer to the recursive definition of $\mathcal{N}_S(u)$. The initial step in the proof of our theorem, therefore, refers to labelled symbols in $\mathcal{N}_S(u)$ which are data dependent on u if $u \in \mathcal{V}$, or which lie in $whilePreds^{(\mathcal{L})}(S)$ if $u = \omega$. This motivates Lemmas 25 and 26.

Lemma 25. *Let S be a schema and assume that there exists a gGv-segment in S for $g \in \mathcal{F}$, $v \in \mathcal{V}$, and $G \in \mathcal{F}^*$. Assume that S is free. Then there is an interpretation i such that g occurs in the term $\mathcal{M}[\![S]\!]^i_e(v)$.*

Proof. Since S is free, there is an interpretation i such that the path $\pi_S i, e$ is terminating and ends in the gGv-segment, and so $\mathcal{M}[\![S]\!]^i_e(v)$ is a gG-term. □

Lemma 26. *Let S be a function-linear schema and assume that $p^{(l)} \in whilePreds^{(\mathcal{L})}(S)$. Assume that S is free. Then there exists a $p^{(l)}\omega$-couple for S.*

Proof. Let μ be an infinite path in S which passes through $p^{(l)}$ and enters the body of $p^{(l)}$ on the first occasion that it does so and does not subsequently pass through $\lessdot p^{(l)} = \mathsf{F} \gtrdot$ or enter the body of any labelled while predicate lying in

the body of $p^{(l)}$. Since S is free, there exists a interpretation i such that $\pi_S i, e = \mu$. We may assume that i maps every predicate term $q(\mathbf{t})$ to F unless either $q(\mathbf{t})$ occurs along μ before μ enters the body of $p^{(l)}$ or $q = p$ and a component of $\mathbf{t}$ is an f-term for $f \in \mathcal{F}$ in the body of $p^{(l)}$. Since S is free, each time the path μ passes through $p^{(l)}$ it defines a different predicate term, and so must have passed through some $f \in \mathcal{F}$ since the previous time, so the predicate terms $p(\mathbf{t})$ defined at $p^{(l)}$ satisfy this condition.

Let $p(\mathbf{t}')$ be the predicate term defined by μ at the first occurrence of $p^{(l)}$ and define the interpretation $j = i(p(\mathbf{t}') = \mathsf{F})$. Every time $\pi_S j, e$ meets a while predicate after the first occurrence of $p^{(l)}$, the predicate term $q(\mathbf{t})$ so defined does not contain any function symbols lying in the body of $p^{(l)}$, since S is function-linear, and thus $q^j(\mathbf{t}) = \mathsf{F}$ and so $\pi_S j, e$ terminates and hence $\{i, j\}$ satisfies the required conditions. □

Lemma 27 shows that the set of all labelled symbols of an f-LFL schema S which are 'involved' in some $p^{(l)}u$-couple for S (that is, $p^{(l)}$ plus the function symbols occurring in the predicate term on which the interpretations in the couple differ) has the same closure property as the set $\mathcal{N}_S(u)$.

Lemma 27. *Let S be an f-LFL schema. Let $u \in \mathcal{V} \cup \{\omega\}$, let $q^{(m)} \in Preds^{(\mathcal{L})}(S)$ and suppose that for $f \in \mathcal{F}$, $F \in \mathcal{F}^*$, there exists an $fFq^{(m)}u$-couple for S. Suppose that $p^{(l)} \searrow_S f$ holds for a labelled predicate $p^{(l)}$ in S. Then there exists a $p^{(l)}u$-couple for S.*

Proof. We will assume that $p^{(l)} \searrow_S f(X)$ holds. Let $I_1 = \{i_1, j_1\}$ be an $fFq^{(m)}u$-couple for S. If $u \in \mathcal{V}$ then we may assume that i_1 maps finitely many predicate terms to T. The q-predicate term at which i_1 and j_1 differ contains a subterm $f(\mathbf{s})$. Let $p(\mathbf{t}_1)$ be the predicate term defined by the prefix $head_S(I_1)$ when it passes through $p^{(l)}$ on the last occasion before defining the term $f(\mathbf{s})$. Define recursively a sequence $I_1 = \{i_1, j_1\}, I_2 = \{i_2, j_2\}, \ldots$ of $f(\mathbf{s})Fqu$-couples for S and predicate terms $p(\mathbf{t}_1), p(\mathbf{t}_2), \ldots$ as follows. Assume that $I_n = \{i_n, j_n\}$ has been defined and $p(\mathbf{t}_n)$ is the predicate term defined by the prefix $head_S(I_n)$ when it passes through $p^{(l)}$ on the last occasion before defining the term $f(\mathbf{s})$. Define $i_{n+1} = i_n(p(\mathbf{t}_n) = \neg X)$ and similarly for j_{n+1} provided

$$\mathcal{M}[\![S]\!]_e^{i_n(p(\mathbf{t}_n)=\neg X)}(u) \neq \mathcal{M}[\![S]\!]_e^{j_n(p(\mathbf{t}_n)=\neg X)}(u)$$

holds; otherwise $I_n = \{i_n, j_n\}$ is the last element in the sequence. Observe that if $u \in \mathcal{V}$ then by the freeness of S and the finiteness assumption on i_1, the paths $\pi_S i_n, e$ and $\pi_S j_n, e$ all terminate.

Suppose that the sequence is finite; that is, the inequality above is false for some minimal n. Hence for some $k \in \{i_n, j_n\}$, $\{k, k(p(\mathbf{t}_n) = \neg X)\}$ is a $p(\mathbf{t}_n)u$-couple for S, whose paths diverge at $p^{(l)}$, as required.

Suppose on the other hand that the sequence is infinite. For each n, define μ_n to be the prefix of $\pi_S i_{n+1}, e)$ up to and including the occurrence of the letter $<p^{(l)} = \neg X>$, at which point $p(\mathbf{t}_n)$ is defined and the paths $\pi_S i_n, e)$ and $\pi_S i_{n+1}, e)$ diverge, with $\pi_S i_n, e)$ passing through $<p^{(l)} = X>$. We now show that the term $f(\mathbf{s})$ is not defined along any μ_n, since any prefix of any such μ_n before the occurrence of $p^{(l)}$ defining $p(\mathbf{t}_n)$ is shared with $\pi_S i_n, e$, which passes through $<p^{(l)} = X>$ and subsequently defines the term $f(\mathbf{s})$, contradicting the liberality of S. Thus $f(\mathbf{s})$ occurs along each path $\pi_S i_{n+1}, e$ *after* $p(\mathbf{t}_n)$ is reached, and since i_{n+1} maps $p(\mathbf{t}_n)$ to $\neg X$, the last occurrence of $p^{(l)}$ before $f(\mathbf{s})$ is defined (which defines $p(\mathbf{t}_{n+1})$) is after $p(\mathbf{t}_n)$ is reached. Thus each prefix μ_n is a strict prefix of μ_{n+1}, and by induction on n, μ_n passes through all the predicate terms $p(\mathbf{t}_1), \ldots, p(\mathbf{t}_n)$ at $p^{(l)}$ in that order, but not (since S is free) through any predicate terms $p(\mathbf{t}_s)$ for $s > n$. We consider two separate cases.

- Assume $u \in \mathcal{V}$. We will show that this leads to a contradiction. Let $K \geq 1$ be an upper bound on the number of predicate terms which i_1 maps to T. If $X = \mathsf{T}$ then i_{K+2} would map negatively many predicate terms to T, giving a contradiction. If $X = \mathsf{F}$ (and so $p^{(l)}$ is an if predicate, since $p^{(l)} \searrow_S f(X)$ holds) then $K + n$ is an upper bound on the number of predicate terms which i_{n+1} maps to T. The prefix μ_n (and hence also the path $\pi_S i_{n+1}, e$) passes at least n times through $<p^{(l)} = \mathsf{T}>$ and hence also passes at least n times through the labelled while predicate lying immediately above the if predicate $p^{(l)}$, defining a predicate term which i_{n+1} maps to T in each case. Since $2n > K + n$ holds if $n > K$, this contradicts the freeness of S.
- Assume $u = \omega$. We may assume that the path $\pi_S i_1, e$ terminates; else interchange i_1 and j_1. Thus there exists $M \geq 2$ such that $\pi_S i_1, e$ does not pass through any predicate term $p(\mathbf{t}_n)$ for $n \geq M$. Define the interpretation $\kappa = i_1(\forall n \geq 1,\ p(\mathbf{t}_n) = \neg X)$ and also the interpretations $k_n = i_n(\forall r \geq M,\ p(\mathbf{t}_r) = \neg X)$.
  For any $n \leq M$ the interpretations κ, i_n and k_n differ only on the predicate terms $p(\mathbf{t}_s)$ for $s \geq n$, through which the prefix μ_n does not pass, hence μ_n is a prefix of the paths they define and so $\pi_S \kappa, e = \pi_S k_M, e$ is infinite.

Clearly $\pi_S k_1, e = \pi_S i_1, e$ which is finite; thus we may define $s < M$ to be maximal such that $\pi_S k_s, e$ is finite. The interpretation k_{s+1} differs from k_s only at $p(\mathbf{t}_s)$, which is defined at $p^{(l)}$ at the end of μ_s, hence $\{k_s, k_{s+1}\}$ is a $p^{(l)}(\mathbf{t}_s)\omega$-couple for S. □

Our main theorems follow. Theorem 28 shows that every element of $\mathcal{N}_S(u)$ for an f-LFL schema S can be detected from the semantics of the schema.

Theorem 28. *Let S be an f-LFL schema and let $u \in \mathcal{V} \cup \{\omega\}$.*

(1) *Let $f \in \mathcal{F} \cap \mathcal{N}_S(u)$; then either there exists an $fFq^{(l)}u$-couple for S for some $F \in \mathcal{F}^*$ and $q^{(l)} \in \mathcal{N}_S(u)$ or $u \in \mathcal{V}$ and f occurs in a term $\mathcal{M}[\![S]\!]^i_e(u)$ for some interpretation i.*

(2) *Let $q^{(l)} \in \mathcal{N}_S(u)$; then there exists a $q^{(l)}u$-couple for S.*

(3) *If T is any (not necessarily free or liberal) u-slice of S, then every symbol in $\mathcal{N}_S(u)$ occurs in T, with the same labels in the case of predicates.*

Proof. Let N be the set of labelled symbols in S for which (1) or (2) holds, according to whether an element of N is a function or a labelled predicate symbol. To prove (1) and (2), we are required to show that $N \supseteq \mathcal{N}_S(u)$. This follows from the following facts.

- The set N contains those elements $f \in \mathcal{F} \cap \mathcal{N}_S(u)$ such that S contains an fFu-segment for some $F \in \mathcal{F}^*$ and $u \in \mathcal{V}$ (since they satisfy (1) by Lemma 25) and also contains the elements of $whilePreds^{(\mathcal{L})}(S)$ if $u = \omega$ (since they satisfy (2) by Lemma 26).
- Let $f \in \mathcal{F}$ and suppose S contains an $fFq^{(l)}$-segment for $f \in \mathcal{F}$, $F \in \mathcal{F}^*$ and a labelled predicate $q^{(l)} \in N$. Then $f \in N$ by Lemma 24.
- Let $p^{(l)} \in Preds^{(\mathcal{L})}(S)$ and suppose $p^{(l)} \searrow_S f$ for $f \in \mathcal{F}$ satisfying (1). If there exists an $fFq^{(l)}u$-couple for S for some $F \in \mathcal{F}^*$ and $q^{(l)} \in \mathcal{N}_S(u)$, then $p^{(l)} \in N$ follows from Lemma 27. Otherwise, $u \in \mathcal{V}$ and there exists an fFu-segment in S for some $F \in \mathcal{F}^*$. In this case, let S' be the schema *S if $q(u)$ then $u := f_1()$; else $u := f_2()$;*, where q, f_1 and f_2 are symbols not occurring in S. Clearly S' is f-LFL, and there exists a qu-couple for S'. Since S' contains an fFq-segment, by Lemma 27 there exists a $p^{(l)}u$-couple for S', and so clearly there exists a $p^{(l)}u$-couple for S; hence $p^{(l)} \in \mathcal{N}_S(u)$.

Thus N contains all the 'initial' labelled symbols that $\mathcal{N}_S(u)$ does, and satisfies the same closure conditions as $\mathcal{N}_S(u)$, thus proving $N \supseteq \mathcal{N}_S(u)$.

We now prove Part (3). Let T be any u-slice of S; then by Proposition 22, every function symbol in $\mathcal{N}_S(u)$ satisfies the conclusion of Part (1) with S replaced by T, proving that T contains every function symbol in $\mathcal{N}_S(u)$. Thus it remains to prove Part (3) for labelled predicates. Let $p^{(l)} \in Preds^{(\mathcal{L})}(S) \cap \mathcal{N}_S(u)$; if $p^{(l)} \searrow_S f$ for some $f \in \mathcal{F} \cap \mathcal{N}_S(u)$, then by the definition of a slice, $p^{(l)} \in Preds^{(\mathcal{L})}(T)$, as required. If $u \in \mathcal{V}$ or $p^{(l)}$ is an if predicate in T, then this condition clearly holds, by the definition of $\mathcal{N}_S(u)$. If on the other hand, $p^{(l)} \in whilePreds^{(\mathcal{L})}(S)$ and $u = \omega$, then $p^{(l)} \searrow_S f$ for some $f \in \mathcal{F} \cap \mathcal{N}_S(u)$ follows from the freeness of S, again showing that $p^{(l)} \in Preds^{(\mathcal{L})}(T)$; hence Part (3) holds. □

The converse of Part (3) of Theorem 28 is false, for if S is the f-LFL schema of Fig. 5, then the slice $T =$ *while $q(w)$ do $w := h(w)$;* is not a u-slice of S if $w \neq u \in \mathcal{V}$, since the interpretation i which maps the predicate term $q(g())$ to F and every other predicate term to T satisfies $\mathcal{M}[\![S]\!]^i_e = \perp \neq \mathcal{M}[\![T]\!]^i_e$. Clearly $\mathcal{N}_S(u) = \emptyset$ and Λ is the minimal u-slice of S.

Theorem 29. *Let S be an f-LFL schema and let $u \in \mathcal{V} \cup \{\omega\}$ and let T be the slice of S satisfying $Symbols^{(\mathcal{L})}(T) = \mathcal{N}_S(u)$. Then T is a u-slice of S, and is the unique minimal u-slice of S (with slices ordered according to their labelled symbol sets).*

$$w := g();$$
$$while\ q(w)\ do\ w := h(w);$$

Fig. 5. Deleting the g-assignment gives a schema which is not a u-slice if $u \neq w$.

```
while q(w) do
        {
        w := h(w);
        if q(w)      then
                        {
                        v := k(v);
                        w := f1(w);
                        }
                     else  w := f2(w);
        }
```

Fig. 6. $Symbols^{(\mathcal{L})}(S) = \mathcal{N}_S(v)$ for this f-LFL schema.

Proof. We have shown in Theorem 20 that T is a u-slice of S. Now let T' be any u-slice of S; then $Symbols^{(\mathcal{L})}(T') \supseteq Symbols^{(\mathcal{L})}(T)$ follows from Part (3) of Theorem 28. □

Theorem 29 does not hold for all schemas; for example, in the schema S of Fig. 4, $f \in \mathcal{N}_S(v)$ holds, but as mentioned in Section 1.2, the slice of S obtained by deleting the f-assignment is a v-slice of S. The Theorem is also false for the schema of Fig. 3.

8. Conclusions and directions for future work

We have shown that for any $u \in \mathcal{V} \cup \{\omega\}$, the minimal u-slice of a function-linear, free, liberal schema S is unique, and is precisely the slice of S containing the set of function symbols and labelled predicate symbols in $\mathcal{N}_S(u)$, and no others. For $u \in \mathcal{V}$, this shows that Weiser's PDG-based static slicing algorithm[2] never gives larger slices than any algorithm which uses just the data and control dependence relations when applied to a program giving rise to a function-linear free liberal schema; thus such an algorithm does not give any advantage over Weiser's even if it 'knows' when predicates occurring in different parts of a program are the same.

Fig. 6 gives an example of a schema S for which $Symbols^{(\mathcal{L})}(S) = \mathcal{N}_S(v)$ holds. By Theorem 12 S is f-LFL (and, in fact, conservative) and hence no strict slice of S is a v-slice. S is not linear, thus showing the strengthening of our main result compared to that of [3].

Further work will 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 not liberal.

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.

Acknowledgement

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

References

[1] M.S. Paterson, Equivalence Problems in a Model of Computation, Ph.D. Thesis, University of Cambridge, UK, 1967.
[2] 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.

[3] S. Danicic, C. Fox, M. Harman, R.M. Hierons, J. Howroyd, M. Laurence, Static program slicing algorithms are minimal for free liberal program schemas, The Computer Journal 48 (6) (2005) 737–748.
[4] F. Tip, A survey of program slicing techniques, Technical Report CS-R9438, Centrum voor Wiskunde en Informatica, Amsterdam, 1994.
[5] D.W. Binkley, K.B. Gallagher, Program slicing, in: M. Zelkowitz (Ed.), Advances in Computing, vol. 43, Academic Press, 1996, pp. 1–50.
[6] S. Danicic, Dataflow Minimal Slicing, Ph.D. Thesis, University of North London, UK, School of Informatics, April 1999.
[7] M. Laurence, S. Danicic, M. Harman, R.M. Hierons, J. Howroyd, Equivalence of linear, free, liberal, structured program schemas is decidable in polynomial time, Theoretical Computer Science, in press.
[8] M.R. Laurence, S. Danicic, M. Harman, R. Hierons, J. Howroyd, Equivalence of linear, free, liberal, structured program schemas is decidable in polynomial time, Technical Report ULCS-04-014, University of Liverpool. <http://www.csc.liv.ac.uk/research/techreports/> 2004.
[9] Z. Manna, Mathematical Theory of Computation, McGraw-Hill, 1974.