Domain Theory

Alexander Vandenbroucke

Domain Theory

Arguing Semantics

Alexander Vandenbroucke

Semantics

Types and Programming Languages

Benjamin C. Pierc

$$\frac{t_1 \rightarrow t_1'}{t_1\ t_2 \rightarrow t_1'\ t_2} \qquad \frac{t_2 \rightarrow t_2'}{v_1\ t_2 \rightarrow v_1\ t_2'}$$

$$\frac{}{(\lambda x.t)\ v \rightarrow [x \mapsto v]t}$$

Operational Semantics

$$\frac{t_1 \to t_1'}{t_1\ t_2 \to t_1'\ t_2} \qquad \frac{t_2 \to t_2'}{v_1\ t_2 \to v_1\ t_2'}$$

$$\frac{}{(\lambda x.t)\ v \to [x \mapsto v]t}$$

Operational Semantics

$$t ::= c \mid t + t$$

$$[\![x + y]\!] = [\![x]\!] + [\![y]\!]$$
$$[\![c]\!] = c$$

$$t ::= c \mid t + t$$

$$[\![x + y]\!] = [\![x]\!] + [\![y]\!]$$
$$[\![c]\!] = c$$

$$[\![.]\!] : \mathrm{Syntax} \to \mathrm{N}$$

$$t ::= c \mid t + t$$

$$[\![x + y]\!] = [\![x]\!] + [\![y]\!]$$
$$[\![c]\!] = c$$

$$[\![.]\!] : \text{Syntax} \rightarrow N$$

Denotational Semantics

$$t ::= c \mid t + t$$

$$[\![x + y]\!] = [\![x]\!] + [\![y]\!]$$

$$[\![c]\!] = c$$

$[\![.]\!]$: Syntax $\rightarrow$ *Math Object*

Denotational Semantics

Why mathematics?

The mathematical objects have structure

$\Rightarrow$ A ***domain*** of objects.

Why mathematics?

The mathematical objects have structure that:

- captures the **meaning**
- allows **reasoning** about it

Why mathematics?

The mathematical objects have structure that:

- captures the **meaning**
- allows **reasoning** about it

$\Rightarrow$ It's all about arguing (about) semantics.

Examples

```
% file prolog.pl:
e(0).
e(X) :- Y is X - 1, o(Y).
o(X) :- Y is X - 1, e(Y).
```

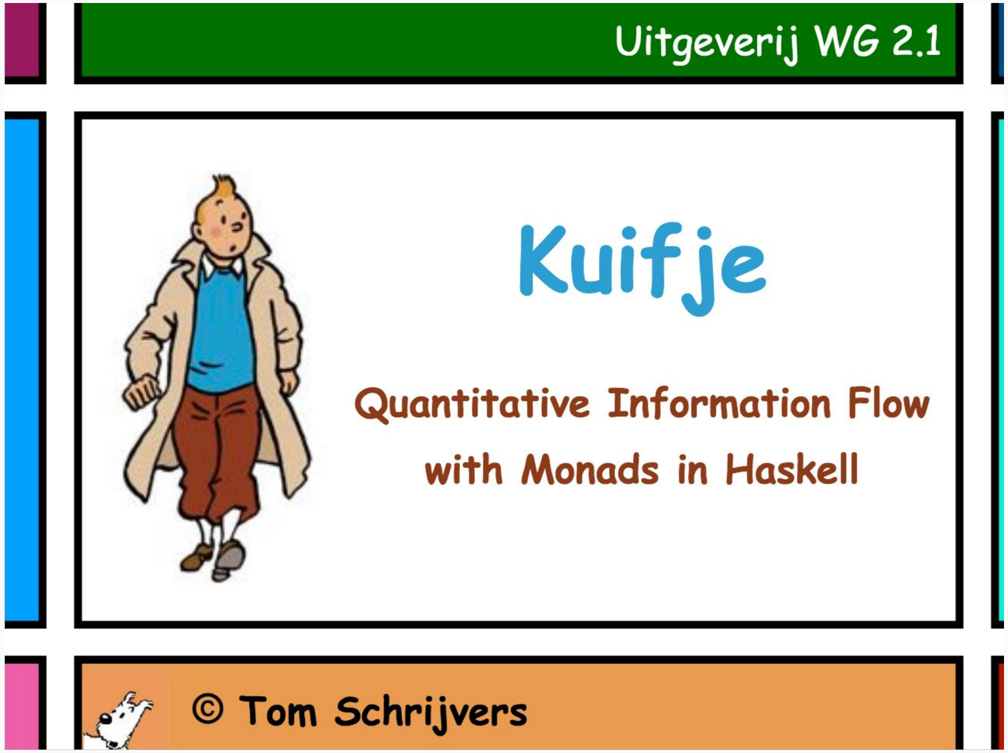
$[\![\texttt{prolog.pl}]\!] = ?$

Examples

```
% file prolog.pl:
e(0).
e(X) :- Y is X - 1, o(Y).
o(X) :- Y is X - 1, e(Y).
```

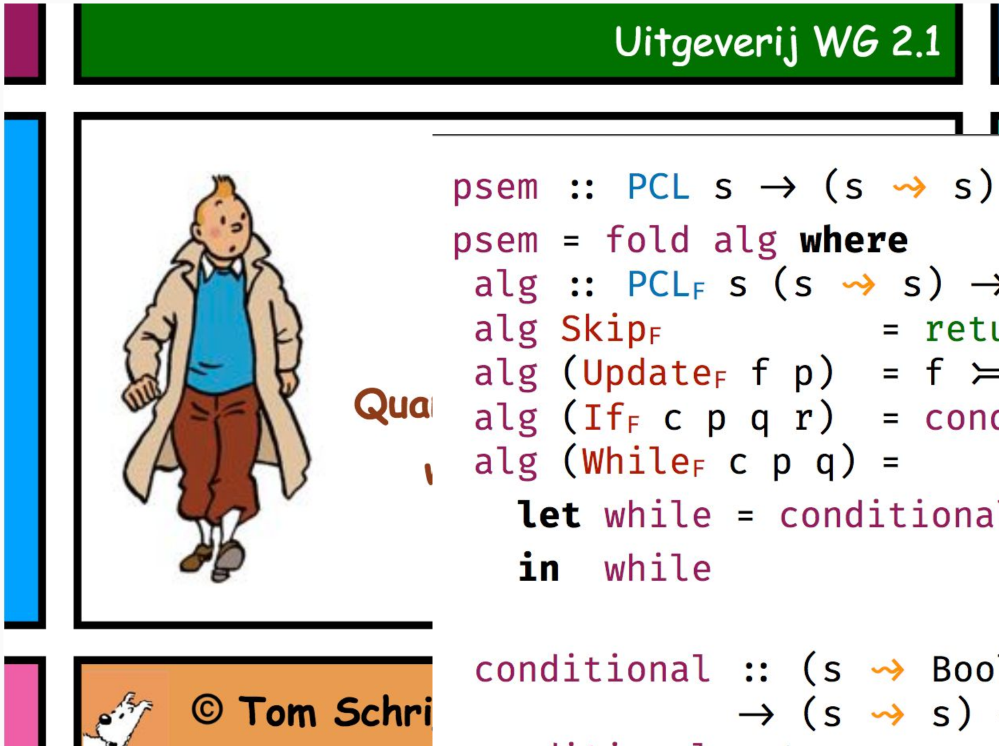
$$[\![\texttt{prolog.pl}]\!] \in P(H)$$

Examples



Tom's Kuifje Talk

Examples



Semantics

```
psem :: PCL s → (s ⇝ s)
psem = fold alg where
 alg :: PCL_F s (s ⇝ s) → (s ⇝ s)
 alg Skip_F            = return
 alg (Update_F f p)    = f ⟹ p
 alg (If_F c p q r)    = conditional c p q ⟹ r
 alg (While_F c p q) =
   let while = conditional c (p ⟹ while) q
   in  while

 conditional :: (s ⇝ Bool) → (s ⇝ s)
             → (s ⇝ s) → (s ⇝ s)
 conditional c t e =
   (c &&& return) ⟹
   (\(b,s) → if b then t s else e s)
```

Tom's Kuifje Talk

Examples

$$t ::= v \mid t\ t \mid \lambda x.t$$

$$[\![t\ t]\!] = ?$$

$$[\![\lambda x.t]\!] = ?$$

Values

=

Functions

Examples

$$t ::= v \mid t\ t \mid \lambda x.t$$

$$[\![t\ t]\!] = ?$$

$$[\![\lambda x.t]\!] = ?$$

$$D = D \to D$$

Domain Equation

Domain theory solves these equations

Overview

I. Introduction
II. Lattices
III. A domain for the UTL

Overview

I. Introduction
II. Lattices
III. A domain for the UTL

Lattices

CONTINUOUS LATTICES

by

Dana Scott

Oxford University Computing Laboratory
Wolfson Building
Parks Road
OX1 3QD

Partially Ordered Set $(D,\sqsubseteq)$:

- *reflexive* $x \sqsubseteq x$
- *transitive*

  $x \sqsubseteq y$ and $y \sqsubseteq z$ implies $x \sqsubseteq z$
- *anti-symmetric*

  $x \sqsubseteq y$ and $y \sqsubseteq x$ implies $x = y$

Least Upper Bound $\sqcup S$ of $S \subseteq D$

- *Upper Bound:*
  $\forall x \in S: x \sqsubseteq \sqcup S$
- *Least* Upper Bound*:*
  $\forall x \in S: x \sqsubseteq z \Rightarrow \sqcup S \sqsubseteq z$

Least Upper Bound $\sqcup S$ of $S \subseteq D$

- *Upper Bound:*
  $\forall x \in S: x \sqsubseteq \sqcup S$
- *Least* Upper Bound*:*
  $\forall x \in S: x \sqsubseteq z \Rightarrow \sqcup S \sqsubseteq z$

$$(\forall x \in S: x \sqsubseteq z) \Leftrightarrow \sqcup S \sqsubseteq z$$

Lattice D

$\sqcup\{x,y\}$ exists for every $x,y \in D$

"tralie" "rooster"

Lattice D

$\sqcup\{x,y\}$ exists for every $x,y \in D$

Complete Lattice D

$\sqcup S$ exists for every subset S

Lattice D

$\sqcup\{x,y\}$ exists for every $x,y \in D$

Complete Lattice D

$\sqcup S$ exists for every subset S

$$\sqcup\emptyset = \bot \sqsubseteq x \sqsubseteq = \top = \sqcup D$$

Lattice D

$\sqcup\{x,y\}$ exists for every $x,y \in D$

Complete Lattice D

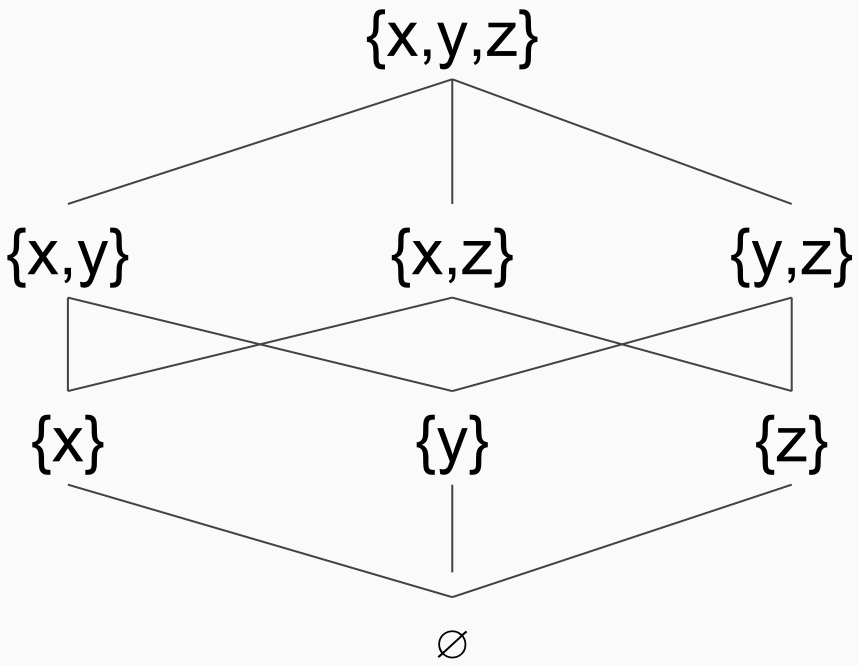
$\sqcup S$ exists for every subset S

Continuous Lattice D

$\forall\ y \in D,\ y = \sqcup S$ for some S

Powerset $(P(S), \subseteq)$

$$P(S) = 2^S = \{X \mid X \subseteq S\}$$



Naturals (N,≤)

Are not complete. Why?

Naturals (N,≤)

Are not complete. Why?

$\sqcup N$ does not exist!

Reals (R,≤)

Are not complete.

Example: lfp semantics for Prolog

$T_P : P(H) \rightarrow P(H)$

$T_P(S) = \{A \mid A \leftarrow B_1,\ldots,B_n \in P,$
$B_1,\ldots,B_n \subseteq S\}$

T_P is monotone

$S_1 \subseteq S_2 \Rightarrow T_P(S_1) \subseteq T_P(S_2)$

Example: lfp semantics for Prolog

$T_P : P(H) \rightarrow P(H)$

$T_P(S) = \{A \mid A \leftarrow B_1,\ldots,B_n \in P,$
$\quad B_1,\ldots,B_n \subseteq S\}$

T_P is monotone

$\Rightarrow$ Thm: lfp(T_P) exists

Example: lfp semantics for Prolog

$T_P : P(H) \rightarrow P(H)$

$T_P(S) = \{A \mid A \leftarrow B_1,\ldots,B_n \in P,$
$B_1,\ldots,B_n \subseteq S\}$

T_P is monotone

$\Rightarrow$ Thm: $\mathrm{lfp}(T_P)$ exists

$\Rightarrow$ $[\![P]\!] = \mathrm{lfp}(T_P)$

Example: lfp semantics for Prolog

```
% file prolog.pl:
e(0).
e(X) :- Y is X - 1, o(Y).
o(X) :- Y is X - 1, e(Y).
```

$[\![\texttt{prolog.pl}]\!] = \sqcup\{T^n(\varnothing)\}$

$= \{e(0), o(1), e(2), o(3), \ldots\}$

Exampl... tics for Pr...

%

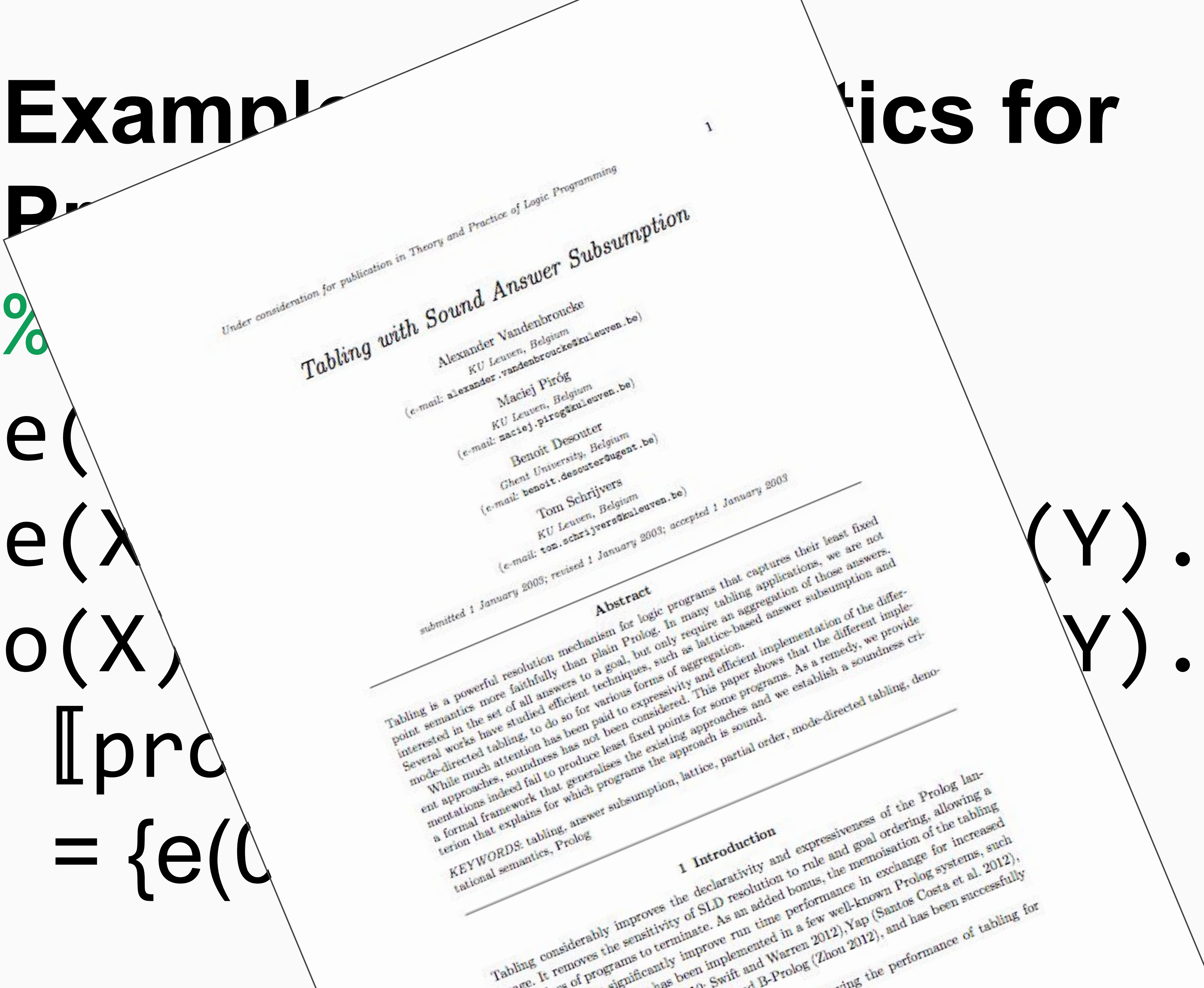
e(

e(X ... (Y).

o(X) ... Y).

⟦pro

= {e(0



1

Under consideration for publication in Theory and Practice of Logic Programming

Tabling with Sound Answer Subsumption

Alexander Vandenbroucke
KU Leuven, Belgium
(e-mail: alexander.vandenbroucke@kuleuven.be)

Maciej Piróg
KU Leuven, Belgium
(e-mail: maciej.pirog@kuleuven.be)

Benoit Desouter
Ghent University, Belgium
(e-mail: benoit.desouter@ugent.be)

Tom Schrijvers
KU Leuven, Belgium
(e-mail: tom.schrijvers@kuleuven.be)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Tabling is a powerful resolution mechanism for logic programs that captures their least fixed point semantics more faithfully than plain Prolog. In many tabling applications, we are not interested in the set of all answers to a goal, but only require an aggregation of those answers. Several works have studied efficient techniques, such as lattice-based answer subsumption and mode-directed tabling, to do so for various forms of aggregation.

While much attention has been paid to expressivity and efficient implementation of the different approaches, soundness has not been considered. This paper shows that the different implementations indeed fail to produce least fixed points for some programs. As a remedy, we provide a formal framework that generalises the existing approaches and we establish a soundness criterion that explains for which programs the approach is sound.

KEYWORDS: tabling, answer subsumption, lattice, partial order, mode-directed tabling, denotational semantics, Prolog

1 Introduction

Tabling considerably improves the declarativity and expressiveness of the Prolog lan-
...age. It removes the sensitivity of SLD resolution to rule and goal ordering, allowing a
...s of programs to terminate. As an added bonus, the memoisation of the tabling
...significantly improve run time performance in exchange for increased
...has been implemented in a few well-known Prolog systems, such
...Swift and Warren 2012),Yap (Santos Costa et al. 2012),
...d B-Prolog (Zhou 2012), and has been successfully
...ing the performance of tabling for

Exampl... ...tics for Pr...

%

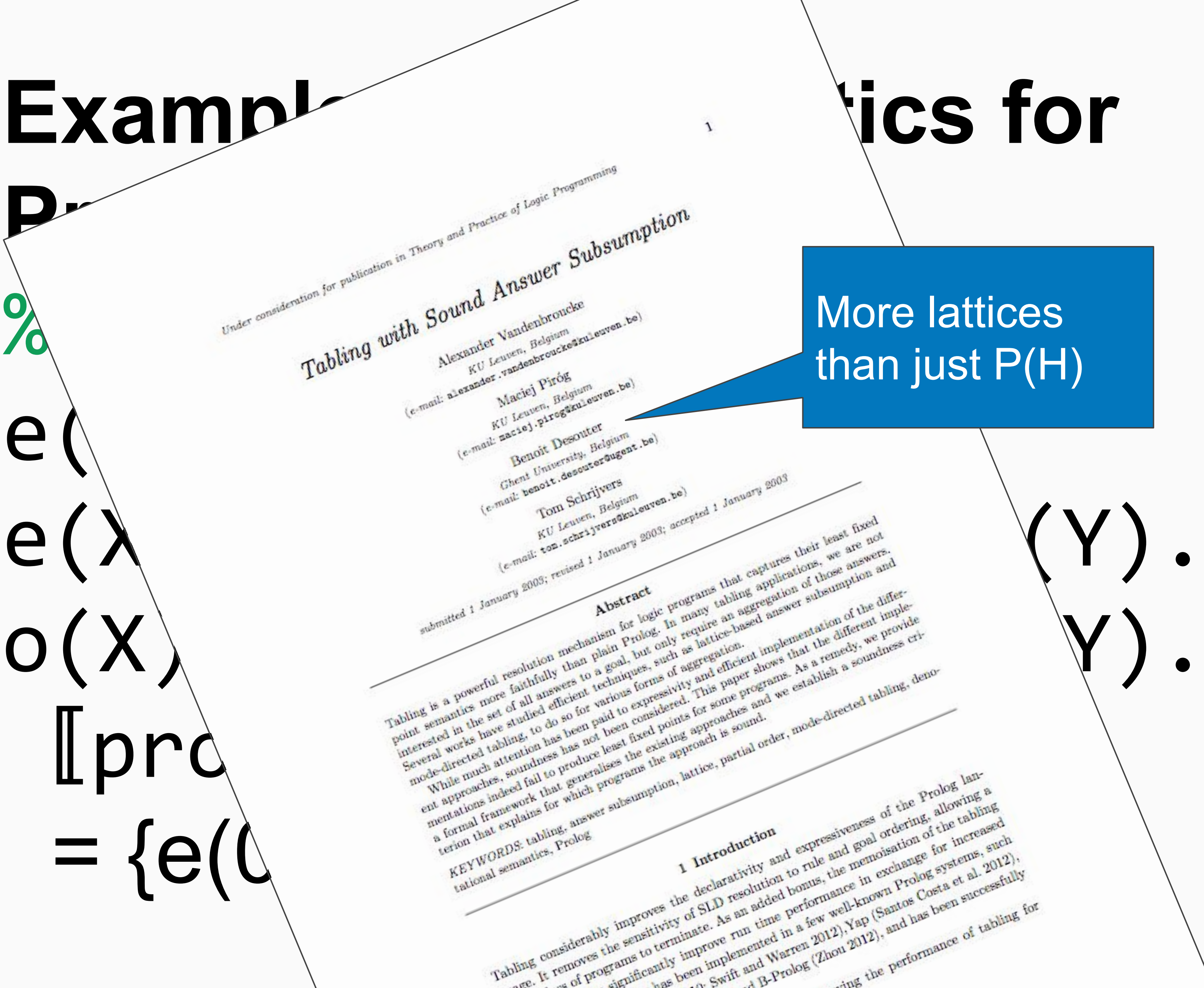
e(

e(X ... (Y).

o(X) ... Y).

⟦pro

= {e((



1

Under consideration for publication in Theory and Practice of Logic Programming

Tabling with Sound Answer Subsumption

Alexander Vandenbroucke
KU Leuven, Belgium
(e-mail: alexander.vandenbroucke@kuleuven.be)
Maciej Piróg
KU Leuven, Belgium
(e-mail: maciej.pirog@kuleuven.be)
Benoit Desouter
Ghent University, Belgium
(e-mail: benoit.desouter@ugent.be)
Tom Schrijvers
KU Leuven, Belgium
(e-mail: tom.schrijvers@kuleuven.be)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Tabling is a powerful resolution mechanism for logic programs that captures their least fixed point semantics more faithfully than plain Prolog. In many tabling applications, we are not interested in the set of all answers to a goal, but only require an aggregation of those answers. Several works have studied efficient techniques, such as lattice-based answer subsumption and mode-directed tabling, to do so for various forms of aggregation. While much attention has been paid to expressivity and efficient implementation of the different approaches, soundness has not been considered. This paper shows that the different implementations indeed fail to produce least fixed points for some programs. As a remedy, we provide a formal framework that generalises the existing approaches and we establish a soundness criterion that explains for which programs the approach is sound.

KEYWORDS: tabling, answer subsumption, lattice, partial order, mode-directed tabling, denotational semantics, Prolog

1 Introduction

Tabling considerably improves the declarativity and expressiveness of the Prolog language. It removes the sensitivity of SLD resolution to rule and goal ordering, allowing a ... of programs to terminate. As an added bonus, the memoisation of the tabling ... significantly improve run time performance in exchange for increased ... has been implemented in a few well-known Prolog systems, such ... Swift and Warren 2012), Yap (Santos Costa et al. 2012), ... B-Prolog (Zhou 2012), and has been successfully ... the performance of tabling for

More lattices than just P(H)

Overview

I. Introduction
II. Lattices
III. A domain for the UTL

Denotational Domain for UTL

$f : D \to D$ is continuous iff

$f(\sqcup S) = \sqcup\{f(x) \mid x \in S\} = \sqcup f(S)$

$([D \to D], \sqsubseteq)$ is continuous

$[D \to D] = \{f : D \to D \mid f \text{ cont} \}$

$f \sqsubseteq g$ iff $f(x) \sqsubseteq g(x)$ for all $x \in D$

An infinite chain of lattices...

$$D_0$$

$$D_1 = [D_0 \to D_0]$$

$$\ldots$$

$$D_{n+1} = [D_n \to D_n]$$

$$\ldots$$

… that we can cast between

$$\mathrm{down}_n : D_{n+1} \to D_n$$
$$\mathrm{down}_n(f) = f(\bot_n)$$

$$\mathrm{up}_n : D_n \to D_{n+1}$$
$$\mathrm{up}_n(x) = \lambda y{:}D_n.x$$

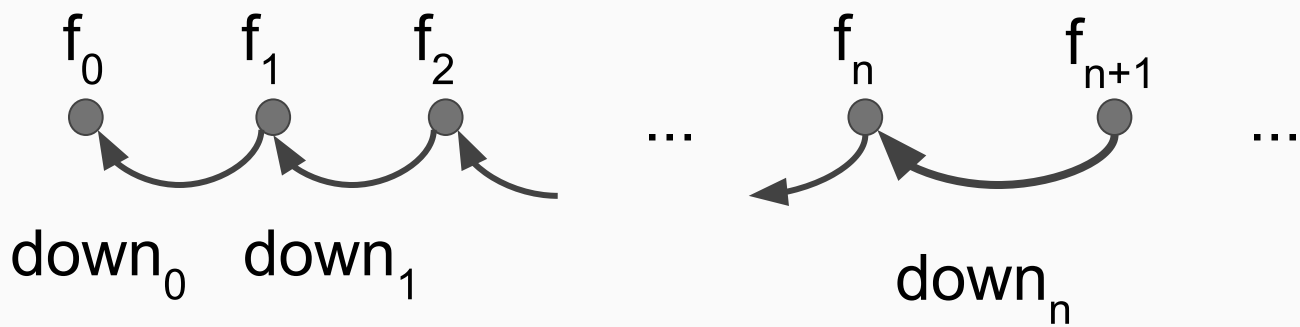
… that we can cast between

$$up_n \;.\; down_n \sqsubseteq id_{n+1}$$
$$down_n \;.\; up_n = id_n$$

(down$_n$ is a projection, and up$_n$ is its inverse)

… now take the limit

$$D_\infty = \{(f_n) \mid f_n \in D_n, f_n = \mathrm{down}_n(f_{n+1})\}$$



… now take the limit

$D_\infty =$
$\{(f_n) \mid f_n \in D_n, f_n = \mathrm{down}_n(f_{n+1})\}$

magic

$\Downarrow$

D_∞ is also a cont. lattice

$x \sqsubseteq y$ iff $x_n \sqsubseteq y_n$ for all n

… now take the limit

$D_\infty =$
$\{(f_n) \mid f_n \in D_n, f_n = down_n(f_{n+1})\}$

~~magic~~ maths

$\Downarrow$

D_∞ is also a cont. lattice

$x \sqsubseteq y$ iff $x_n \sqsubseteq y_n$ for all n

up and down to the limit

$\mathrm{down}_{\infty n} : D_\infty \rightarrow D_n$

$\mathrm{down}_{\infty n}(f) = f_n$

$\mathrm{up}_{n\infty} : D_n \rightarrow D_\infty$

$\mathrm{up}_{n\infty}(x) = (f_m)$ s.t.

$$
\begin{aligned}
f_m \mid m < n &= \mathrm{down}_m(f_{m+1}) \\
\mid m = n &= x \\
\mid m > n &= \mathrm{up}_{m-1}(f_{m-1})
\end{aligned}
$$

D_∞ is its own function space

$$\mathrm{down}_\infty : [D_\infty \to D_\infty] \to D_\infty$$
$$\mathrm{up}_\infty : D_\infty \to [D_\infty \to D_\infty]$$

are inverses, s.t.

$$D_\infty \cong [D_\infty \to D_\infty]$$

Application

$$[\![t_1\ t_2]\!] = \bigsqcup_{n=0\ldots\infty} \underbrace{up_{n\infty}(\overbrace{[\![t_1]\!]_{n+1}}^{D_{n+1}}(\overbrace{[\![t_2]\!]_{n}}^{D_n}))}_{D_\infty}$$

Abstraction

$$[\![\lambda x.t]\!] = \underbrace{\mathrm{down}_\infty(\underbrace{\lambda x' : D_\infty.\ [\![t[x \to x']]\!]}_{D_\infty \to D_\infty})}_{D_\infty}$$

Conclusion

Conclusion

Domain Theory solves domain equations

$$D = D \to D$$

$$V = T + [V \times V] + [V + V] + [V \to V]$$

giving a *mathematical* language for arguing about the **denotational** semantics of programs