Inferenza di tipi (polimorfi) in linguaggi funzionali à la ML

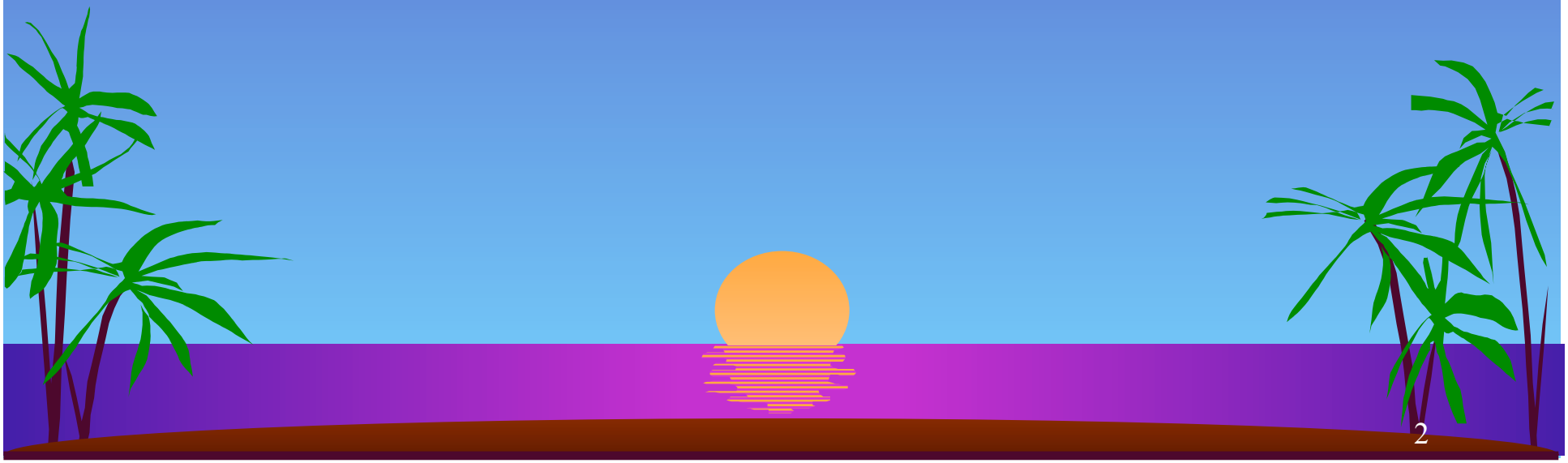
- l'algoritmo di Damas-Milner
  - basato sul sistema di tipo DM
  - formalizzazione elegante, sostenuta da teoremi di correttezza, universalmente accettata
- l'approccio che prevede di assegnare una semantica solo ai programmi ben-tipati (secondo il sistema di tipo) introduce approssimazione
  - alcuni programmi corretti sui tipi (guardando la semantica) non vengono tipati
- la derivazione sistematica via interpretazione astratta
  - fornisce una migliore comprensione sulla relazione con la semantica
  - mostra strade che possono essere tentate per migliorare la precisione

Controesempi

- espressioni che hanno una valutazione concreta senza errori di tipo
  - con la nostra semantica del linguaggio didattico che non ha il type checking a priori

  e che non vengono tipate dall'algoritmo DM

  – la sintassi astratta usata è una piccola variante di quella usata all'inizio del corso
  – c'e' anche la mutua ricorsione



Esempio 1: mostro di Cousot?

- f f1 g n x = $g(f1^n(x))$

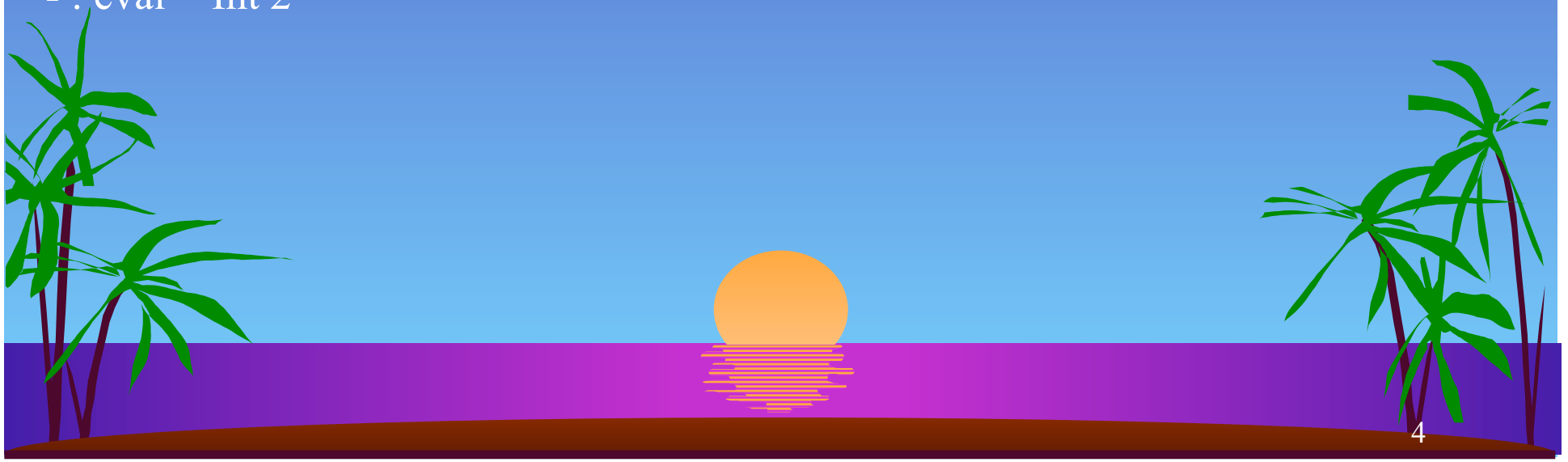
```
# let rec f f1 g n x = if n=0 then g(x) else f(f1) (function x -> (function h -> g(h(x)) )) (n-1) x f1
        in f (function x -> x+1) (function x -> -x) 10 5;;
This expression has type ('a -> 'a) -> 'b  but is here used with type 'b
```

```
# sem (Letrec (Id "f",Fun(Id "f1", Fun(Id "g", Fun(Id "n", Fun(Id "x",
    Ifthenelse(Var(Id "n"),Appl(Var(Id "g"),Var(Id "x")),
    Appl(Appl(Appl(Appl(Appl(Var(Id "f"),Var(Id "f1")),
        Fun(Id "x",Fun(Id "h", Appl(Var(Id "g"),Appl(Var(Id "h"),Var(Id "x"))))))),
         Diff(Var(Id "n"),Eint 1)),Var(Id "x")),Var(Id "f1")))))),
          Appl(Appl(Appl(Appl(Var(Id "f"),Fun(Id "x",Sum(Var(Id "x"),Eint 1))),
           Fun(Id "x",Var(Id "x"))), Eint 10),Eint 5) )   ) emptyenv;;
- : eval = Int 15
```

Esempio 2: ricorsione mutua?

```
# let rec f x = x and g x = f (1+x) in f f 2;;
This expression has type int -> int  but is here used with type int
```

```
# sem (Letmutrec((Id "f",Fun(Id "x",Var(Id "x"))),
          (Id "g",Fun(Id "x",Appl(Var(Id "f"),Sum(Eint 1,Var(Id "x"))))),
        Appl(Appl(Var(Id "f"),Var(Id "f")),Eint 2))) emptyenv;;
- : eval = Int 2
```



Esempio 3: ricorsione polimorfa

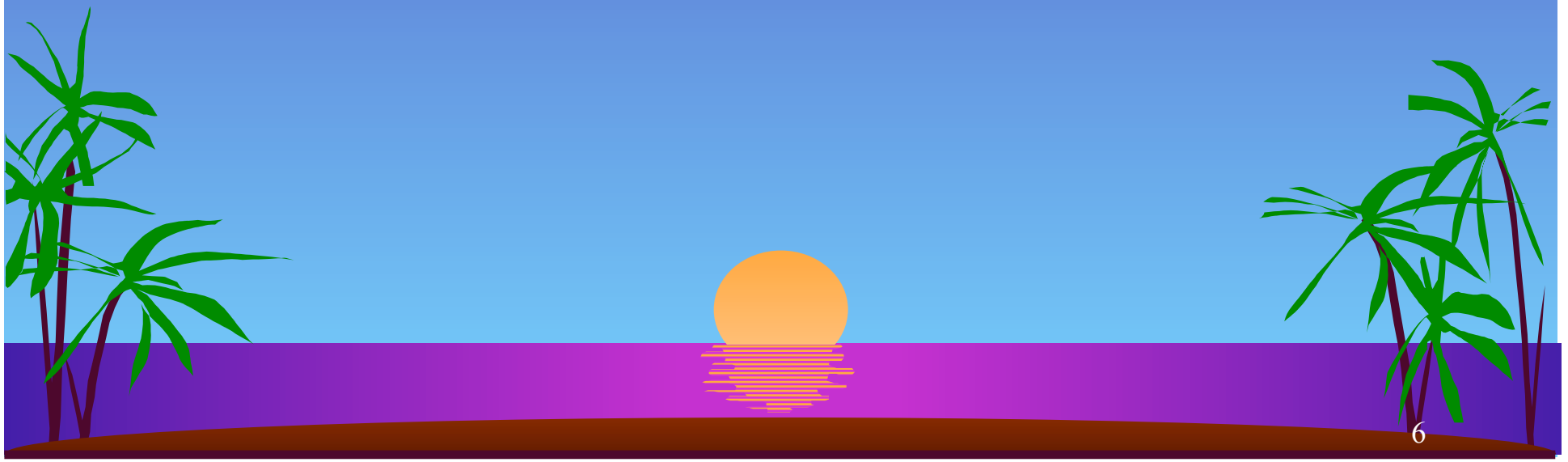
```
# let rec polyf x y = if x=0 then 0 else
                      if (x-1)=0 then (polyf (x-1)) (function z -> z)
                     else (polyf (x-2)) 0 in polyf 3 1;;
This expression has type int but is here used with type ‘a -> ‘a
```

```
# sem (Letrec (Id "polyf", Fun(Id "x", Fun (Id "y",
        Ifthenelse(Var (Id "x"), Eint 0,
         Ifthenelse (Diff (Var (Id "x"), Eint 1), Appl(Appl (Var (Id "polyf"),
          Diff (Var (Id "x"), Eint 1)), Fun (Id "z", Var (Id "z"))),
           Appl (Appl (Var (Id "polyf"), Diff (Var (Id "x"), Eint 2)), Eint 0))))),
        Appl(Appl(Var (Id "polyf"),Eint 3),Eint 1) )) emptyenv;;
- : eval = Int 0
```

Esempio 4: astrazione polimorfa

```
# (function x -> x x) (function x -> x) 3
This expression has type ‘a -> ‘b but is here used with type ‘a
```

```
# sem (Appl(Appl(Fun(Id "x", Appl(Var(Id "x"),Var(Id "x"))),
      (Fun(Id "x",Var(Id "x")))), Eint 3) ) emptyenv;;
- : eval = Int 3
```



Interpretazione astratta e sistemi di tipo

- Patrick Cousot ha ricostruito tutti i sistemi di tipo noti
  - ed i relativi algoritmi di inferenza, ove possibile
  - più alcuni nuovi

  come una gerarchia di interpretazioni astratte (Cousot, POPL 97)
- esistono sistemi di tipo progettati per modellare altre analisi statiche (strictness, varie proprietà collegate alla sicurezza)

- in generale, ogni sistema di tipo deve essere dimostrato corretto rispetto ad una semantica
  - ed il relativo algoritmo di inferenza deve essere dimostrato corretto e completo rispetto al sistema di tipo
- le semantiche astratte sono invece corrette per costruzione, essendo direttamente derivate dalla semantica

Un esperimento utile per confrontare i due approcci nell'inferenza di tipo in ML

- dalla parte dei sistemi di tipo abbiamo
  - i tipi monomorfi di Church-Curry
    - l'algoritmo di inferenza di Hindley
  - i tipi let-polimorfi à la ML, con ricorsione monomorfa
    - l'algoritmo di Damas-Milner
  - la ricorsione polimorfa

- dalla parte dell'interpretazione astratta
  - la ricostruzione da parte di Monsuez [FSTTCS 92, SAS 93] dell'algoritmo di Damas-Milner come calcolo di una semantica astratta
  - la già citata ricostruzione come derivazione sistematica via interpretazione astratta [POPL 97] di una gerarchia di sistemi di tipo ed algoritmi di inferenza.

Fondamenti dell'esperimento 1

- la semantica concreta è la versione collecting della semantica denotazionale del λ-calcolo eager non tipato, senza `let` (esteso con la ricorsione mutua), presa dal pluricitato articolo di Cousot
- la implementazione (in OCaML) della semantica astratta già vista,

  http://www.di.unipi.it/~levi/typesav/pagina2.html

è essenzialmente l'implementazione dell'algoritmo di Damas Milner proposta nel libro di Cousineau-Mauny 98

- la trasmissione di vincoli sulle variabili di tipo (originariamente realizzata con l'ambiente di tipi) è lì realizzata aggiungendo ai monotipi con variabili sostituzioni idempotenti
  - il "dominio astratto" è quello dell'algoritmo di Hindley
    - termini (monotipi con variabili) $*$ sostituzioni idempotenti
    - l'ordinamento parziale sul dominio astratto è basato sulla relazione di istanziazione sui termini
      - il dominio astratto non è Noetheriano

Fondamenti dell'esperimento 2

- la semantica astratta corrispondente all'algoritmo di Damas-Milner
  - la stessa dell'algoritmo di Hindley con tipi monomorfi nel frammento senza `let`
- è ottenuta da Cousot 97 astraendo una semantica con ricorsione polimorfa (à la Mycroft)
- questa astrazione rimuove il calcolo di punto fisso nella semantica astratta delle funzioni ricorsive
  - poiché non c'è calcolo di punto fisso nell'algoritmo DM
- naturalmente il calcolo di punto fisso potrebbe divergere, dato che il dominio non è Noetheriano
- Monsuez ha dimostrato che il trattamento delle funzioni ricorsive nell'algoritmo DM può essere visto come applicazione di un operatore di widening
  - più o meno, un passo di calcolo iterativo seguito da una unificazione

Il widening dell'algoritmo DM

- la prima approssimazione del valore astratto è il bottom del dominio astratto
- prende i risultati delle prime due iterazioni e calcola il loro lub (most general common instantiation, calcolata con l'unificazione)
- se il lub è il top (l'unificazione fallisce), l'espressione non è tipabile (errore di tipo)
- è esattamente un operatore di widening, che calcola una approssimazione superiore (corretta) del lfp

Una idea per migliorare la precisione

- facciamo al massimo k iterazioni del calcolo del punto fisso
- se raggiungiamo un punto fisso, siamo contenti
- altrimenti, applichiamo il widening di Milner alle ultime due approssimazioni Milner's widening to the last two results

- SORPRESA!
- riusciamo a tipare più funzioni
- otteniamo tipi più precisi

L'esempio che ci ha convinto a insistere

- mostro di Cousot
- CaML
  - `# let rec f f1 g n x = if n=0 then (g x) else`

```
                    (((((f f1)(fun x -> (fun h -> (g(h x))))(n - 1))(x))(f1));;
This expression has type ('a -> 'a) -> 'b but is here used with type 'b
```

- il tipo da noi inferito (rappresentato con una sintassi ML)
  - `val f : ('a -> 'a) -> ('a -> 'b) -> int -> 'a -> 'b = <fun>`
  - punto fisso raggiunto in 3 iterazioni

Calcolo del punto fisso astratto

```
let makefunrec (i, e1, r) =
          alfp ((newvar(),[]), i, e1, r  )

let rec alfp (ff, i, e1, r   ) =
         let tnext = funzionale ff i e1 r  in
        (match tnext with
          |(Notype, _) -> (Notype,[])
          |_ -> if abstreq(tnext,ff) then ff
                      else      alfp(tnext, i, e1, r    ) ) )
```

- esistono catene crescenti infinite
- il calcolo del punto fisso può divergere
- abbiamo bisogno di un widening

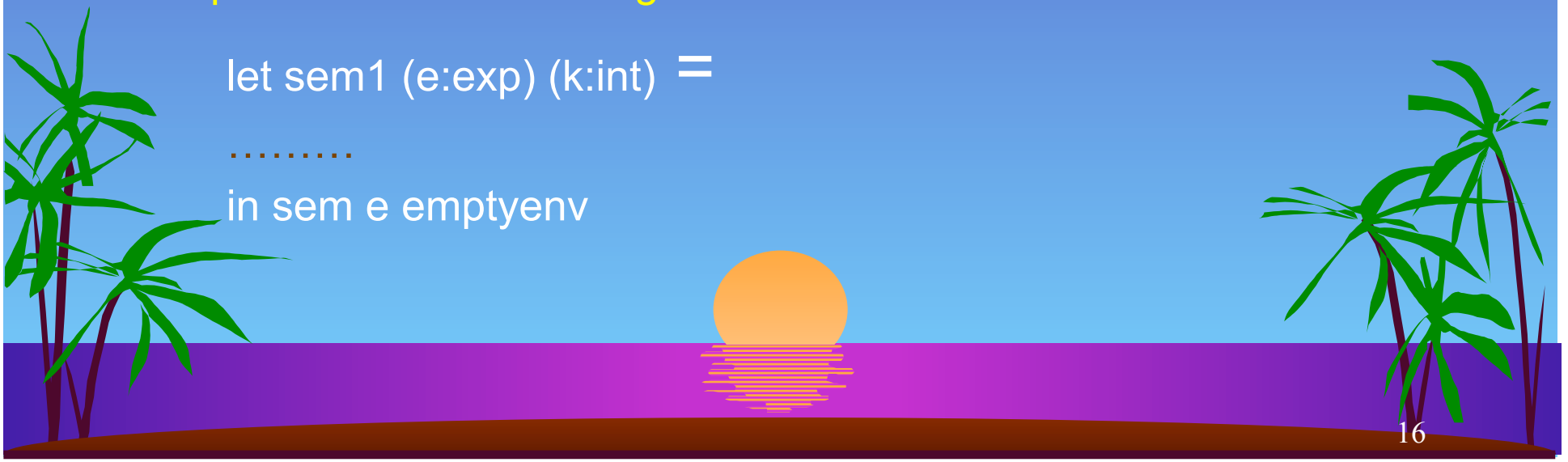
Il nuovo widening

```
let makefunrec (i, e1, r) =
          alfp ((newvar(),[]), i, e1, r, k)

let rec alfp (ff , i, e1, r, n) =
         let tnext = funzionale ff i e1 r  in
        (match tnext with
          |(Notype, _) -> (Notype,[])
          |_ -> if abstreq(tnext,ff) then ff
                        else
                  (if n = 0 then
                  widening(ff,tnext)
                  else    alfp(tnext, i, e1, r, n-1) ) )
let  widening ((f1,c1),(t,c)) =
     let sigma = unifylist( (t,f1) :: (c@c1)) in
     (match sigma with
           |Fail -> (Notype,[])
           |Subst(s) -> (applysubst sigma t,s))
```

Calcolo del punto fisso astratto con widening

- abstreq controlla la varianza dei due termini nei valori astratti
- la funzione di valutazione semantica astratta non viene modificata
  - aggiungiamo una funzione esterna con cui fissiamo il parametro del widening k

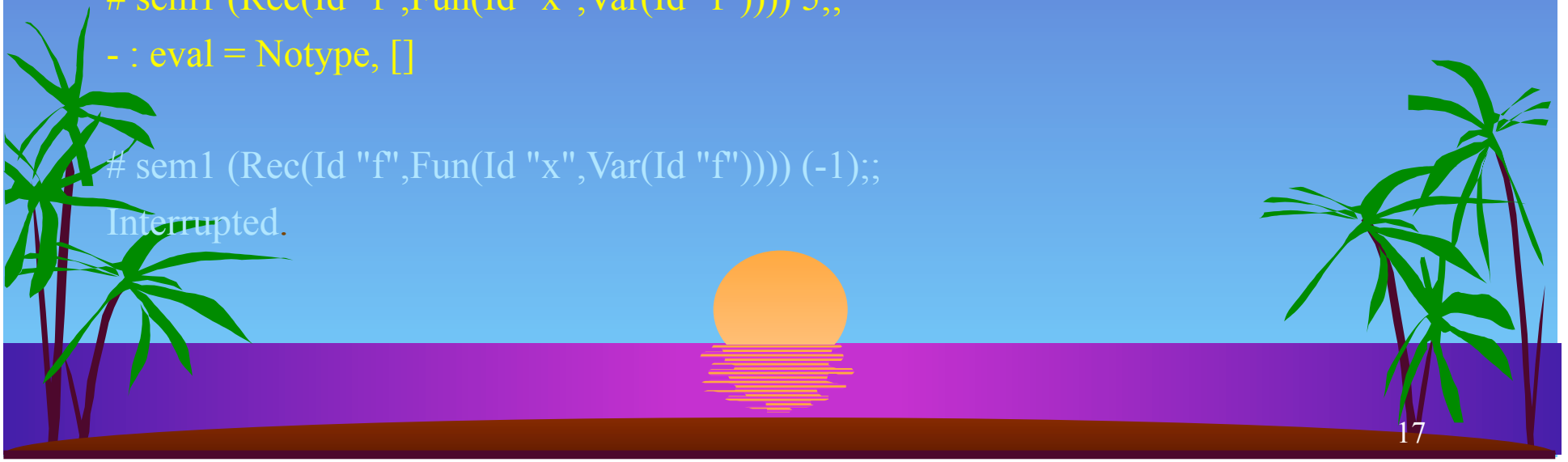
```
let sem1 (e:exp) (k:int) =
.....
in sem e emptyenv
```



Esempio 1: non terminazione

```
# let rec f x = f in f;;
This expression has type 'a -> 'b but is here used with type 'b
```

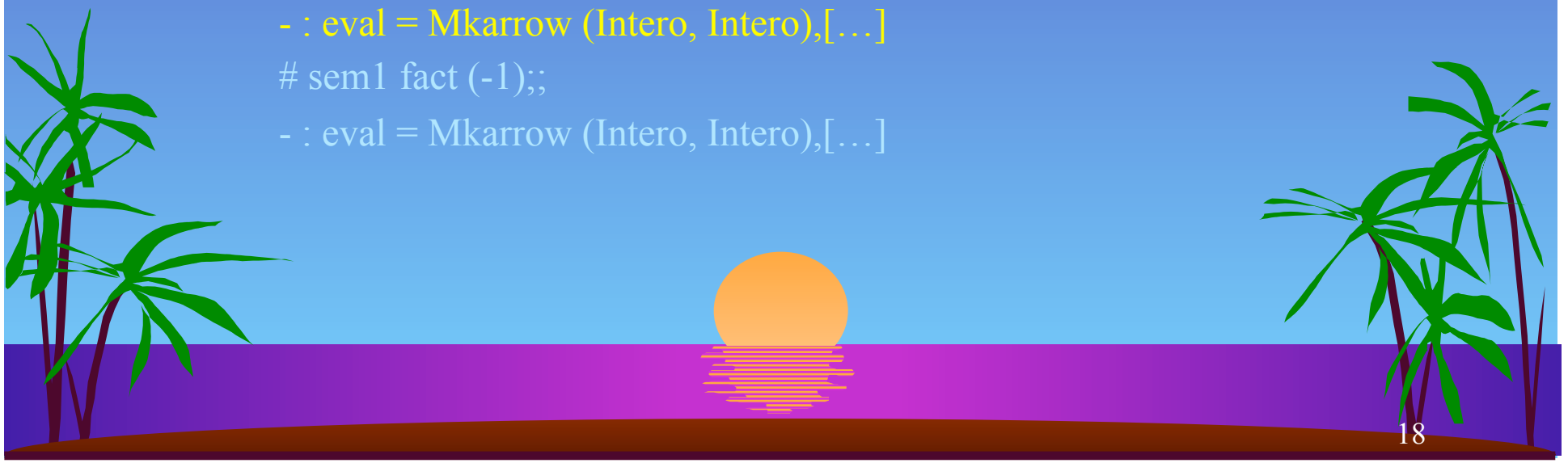
```
# sem1 (Rec(Id "f",Fun(Id "x",Var(Id "f")))) 0;;
- : eval = Notype, []

# sem1 (Rec(Id "f",Fun(Id "x",Var(Id "f")))) 5;;
- : eval = Notype, []

# sem1 (Rec(Id "f",Fun(Id "x",Var(Id "f")))) (-1);;
Interrupted.
```



2: ricorsione “facile”

```
#  let fact =
       Rec ( Id("pfact"), Fun(Id("x"),
           Ifthenelse(Diff(Var(Id("x")),Eint(1)), Eint(1),
           Sum(Var(Id("x")),
               Appl(Var(Id("pfact")),Diff(Var(Id("x")),Eint(1)))))))
       …..
```

```
# sem1 fact 0;;
- : eval = Mkarrow (Intero, Intero),[…]
# sem1 fact (-1);;
- : eval = Mkarrow (Intero, Intero),[…]
```



3: mostro di Cousot

```
# let rec f f1 g n x = if n=0 then g(x)
                       else f(f1)(function x -> (function h -> g(h(x))))
                    (n-1) x f1;;
This expression has type ('a -> 'a) -> 'b but is here used with type 'b
```

```
# let monster = Rec (Id "f",Fun(Id "f1", Fun(Id "g", Fun(Id "n", Fun(Id "x",
      Ifthenelse(Var(Id "n"),Appl(Var(Id "g"),Var(Id "x")),
      Appl(Appl(Appl(Appl(Appl(Var(Id "f"),Var(Id "f1")),
          Fun(Id "x",Fun(Id "h", Appl(Var(Id "g"),Appl(Var(Id "h"),Var(Id
          "x"))))))),Diff(Var(Id "n"),Eint 1)),Var(Id "x")),Var(Id "f1")))))));;
```

```
.....
# sem1 monster 0;;
- : eval = Notype, []
# sem1 monster 1;;
- : eval = Notype, []
```

```
# sem1 monster 2;;
- : eval = Mkarrow (Mkarrow (Vvar "var43", Vvar "var43"),
      Mkarrow  (Mkarrow (Vvar "var43", Vvar "var36"),
      Mkarrow (Intero, Mkarrow (Vvar "var43", Vvar "var36"))),
          [...]
```

- stesso risultato per k=-1 (lfp)

3: Il widening può far perdere precisione

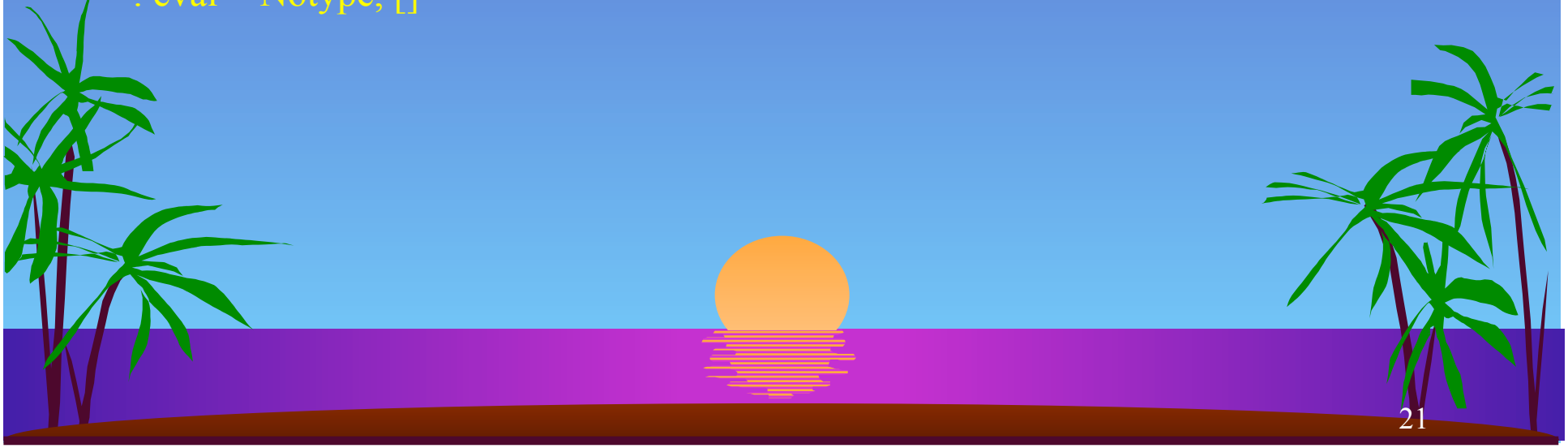
```
# let f1 x = x in let g x = f1 (1+x) in f1;;
- : 'a -> 'a = <fun>
# let rec f x = x and g x = f (1+x) in f;;
- : int -> int = <fun>
```

```
#  let f1 = Let(Id "f",Fun(Id "x",Var(Id "x")), Let(Id "g",Fun(Id "x",Appl(Var(Id "f"),Su
     m(Eint 1,Var(Id "x")))), Var(Id "f")));;
        ....
# sem1 f1 (-1);;
- : eval = Mkarrow (Vvar "var1", Vvar "var1"), [.....]
# let f = Letmutrec((Id "f",Fun(Id "x",Var(Id "x"))),
       (Id "g",Fun(Id "x",Appl(Var(Id "f"),Sum(Eint 1,Var(Id "x"))))), Var(Id "f"));;
        ...
# sem1 f (-1);;
- : eval = Mkarrow (Vvar "var9", Vvar "var9"), [.....]
# sem1 f 0;;
- : eval = Mkarrow (Intero, Intero), [.....]
```

4: niente let polimorfismo

```
# let f x = x in f f;;
- : '_a -> '_a = <fun>
```

```
# sem1 (Let(Id "f",Fun(Id "x", Var(Id "x")),Appl(Var(Id "f"),Var(Id "f")))) 0;;
- : eval = Notype, []
```



I RISULTATI

- per k=1 otteniamo esattamente il risultato di ML
- per ogni k > 1, o raggiungiamo il minimo punto fisso
  - in generale più preciso di ML
- oppure abbiamo esattamente lo stesso risultato
- abbiamo un miglioramento della precisione solo se raggiungiamo il minimo punto fisso in k> 1 passi

- rivediamo alcuni esempi (con la sintassi ML)

Il mostro di Cousot

f f1 g n x = $g(f1^n(x))$

```
# let rec f f1 g n x =
if n=0 then g(x) else f(f1)(function x ->
(function h -> g(h(x)))) (n-1) x f1;;
This expression has type ('a -> 'a) -> 'b but is here used with
 type 'b.
```

- riusciamo a tiparla con due iterazioni in più

```
val f:('a-> 'a) -> ('a-> 'b) -> int-> 'a-> 'b = <fun>
```

- il tipo calcolato è il minimo punto fisso
- Cousot calcola lo stessotipo con un sistema politypo á la Church-Curry

TRACCIA DEL CALCOLO 1

passo 0

τ_0 ='a1 $\quad\quad$ $\gamma_0 = \varepsilon$

passo 1

τ_1 ='a5->('a4->'a2)->int->'a4->'a2

γ_1 = {'a1='a5->('a3->(('a3->'a4)->'a2))->int->'a4 ->('a5->'a2)}

l'unificazione nel widening fallisce

{'a4='a3, 'a2=('a3->'a3)->'a2, …}

This expression has type ('a -> 'a) -> 'b but is here used with type 'b.

TRACCIA DEL CALCOLO 2

passo 1

τ_1 ='a5->('a4->'a2)->int->'a4->'a2

γ_1 = {'a1='a5->('a3->(('a3->'a4)->'a2))->int->'a4 ->('a5->'a2)}

passo 2

τ_2 = ('a7->'a7)->('a7->'a6)->int->'a7->'a6

γ_2 = {'a2=('a7->'a7)->'a6}

l'unificazione nel widening fallisce

{'a5='a7->'a7, 'a4='a7, 'a6=('a7->'a7)->'a6, …}

TRACCIA DEL CALCOLO 3

passo 2

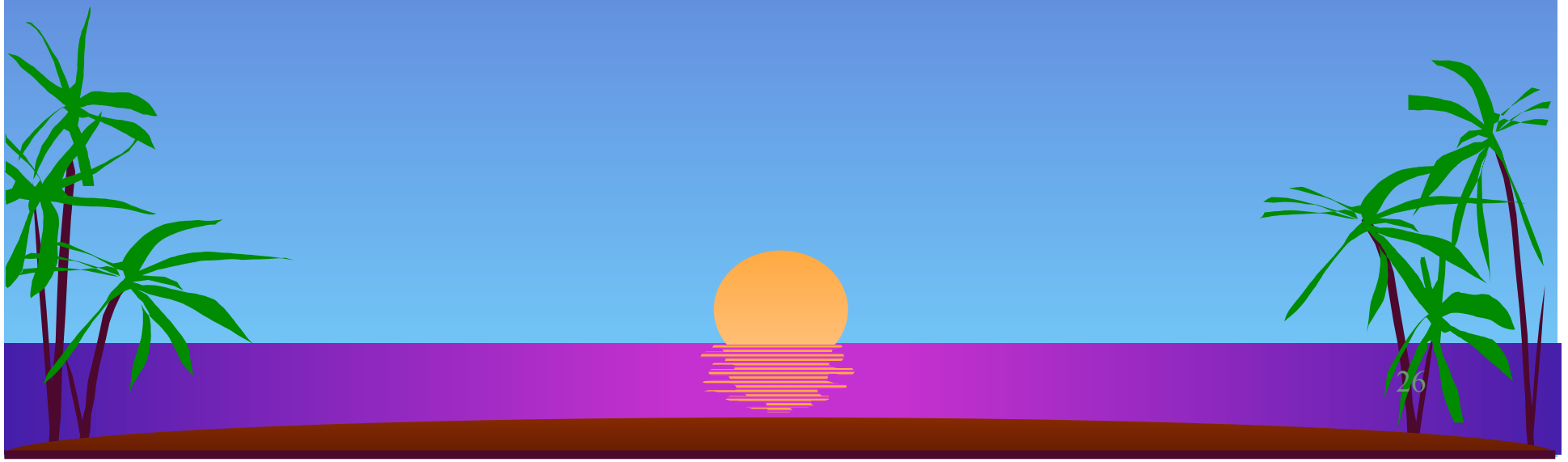
τ_2 = ('a7->'a7)->('a7->'a6)->int->'a7->'a6

γ_2 = {'a2=('a7->'a7)->'a6}

passo 3 (punto fisso)

τ_3 =('a->'a)->('a->'b)->int->'a->'b

γ_3 = {'a6=('a->'a)->'b, 'a7='a }



ESEMPIO 1 di Monsuez

- il risultato di ML

```
# let rec f x = (function x -> function y -> x) 0 (f 0);;
val f: int -> int = <fun>
```

- tipo più preciso con due iterazioni in più

```
val f: 'a ->  int = <fun>
```

- Monsuez trova lo stesso tipo con l'algoritmo per la ricorsione polimorfa
  - ma qui non c'è ricorsione polimorfa!

ESEMPIO 2 di Monsuez

- il risultato di ML
-
```
# let rec p x = if q x = q 1 then p x else p x
      and q x = p 1;;
val p: int -> 'a = <fun>
val q: int -> 'a = <fun>
```

- tipo più preciso con due iterazioni in più

```
val p: 'a -> 'b = <fun>
val q: 'a -> 'b = <fun>
```

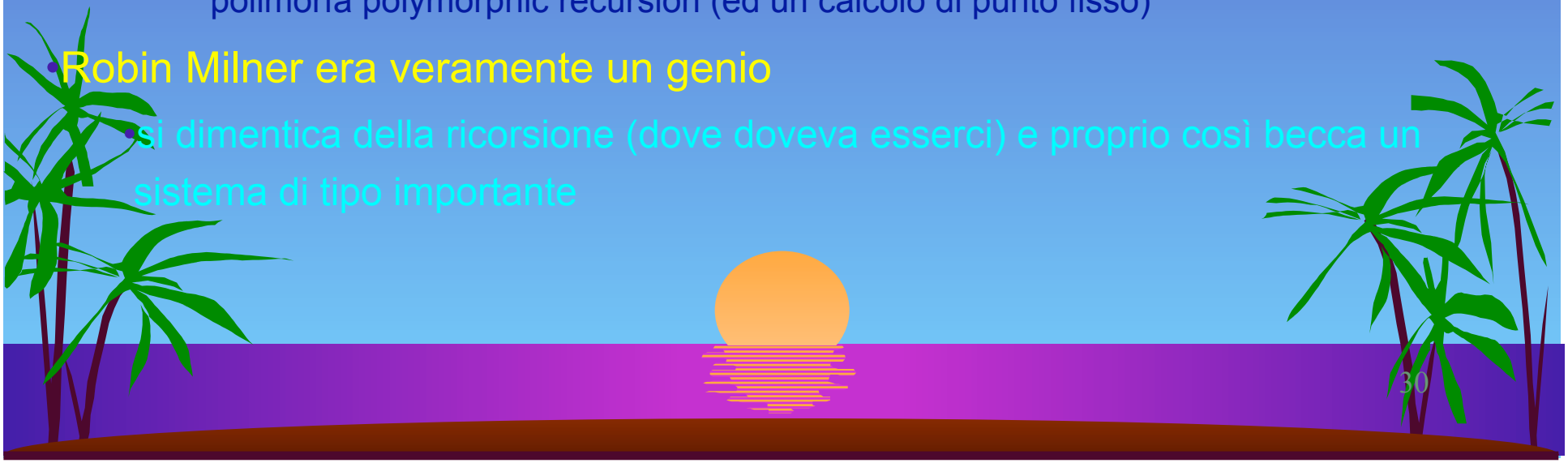
- Monsuez trova lo stesso tipo con l'algoritmo per la ricorsione polimorfa
  - ma qui non c'è ricorsione polimorfa!

DISCUSSIONE

- la nostra semantica astratta sta tra quella di Damas-Milner e quella di Mycroft nella gerarchia di Cousot
- dal punto di vista del sistema di tipo non è chiaro cosa possa esserci tra monomorfismo e ricorsione polimorfa
- l'algoritmo è molto più semplice di quello della ricorsione polimorfa
  - non c'è bisogno di tipi quantificati nell'ambiente
  - di conseguenza non c'è generalizzazione del tipo delle chiamate ricorsive
- ciò malgrado, riesce a tipare tutti gli esempi usati in letteratura a sostegno dell'utilità della ricorsione polimorfa

COSA ABBIAMO IMPARATO DALL'ESPERIMENTO

- calcolando migliori approssimazioni dei punti fissi (iterazione limitata + widening) nella semantica astratta delle funzioni ricorsive
  - riusciamo a inferire tipi più precisi
  - risolviamo alcuni problemi dell'algortmo di inferenza di ML senza ricorrere a nozioni di tipo più complesse
    - mostro di Cousot da lui tipato con un sistema di politipi à la Church-Curry (e con un calcolo di punto fisso nella semantica astratta)
    - i due esempi di Monsuez vengono da lui (e da Mycroft) tipati usando la ricorsione polimorfa polymorphic recursion (ed un calcolo di punto fisso)
- Robin Milner era veramente un genio
  - si dimentica della ricorsione (dove doveva esserci) e proprio così becca un sistema di tipo importante



DAI SISTEMI DI TIPO ALL'INFERENZA DI TIPO

- i sistemi di tipo sono uno strumento molto importante per gestire un'ampia classe di proprietà
  - soprattutto nei linguaggi funzionali ed in quelli ad oggetti
  - ma anche nei calcoli per formalizzare concorrenza e mobilità
- il sistema di tipo riflette direttamente le proprietà a cui siamo interessati
- le regole dei sistemi di tipo sono quasi sempre facili da capire
- è invece spesso moto difficile passare dalle regole del sistema di tipo all'algoritmo di inferenza
  - c'è bisogno di tecniche sistematiche (non siamo tutti Milner!)
  - l'interpretazione astratta fornisce alcune di queste tecniche

L'INTERPRETAZIONE ASTRATTA PUÒ AIUTARE

- quale informazione dobbiamo aggiungere ai tipi (del sistema di tipo) per permettere una inferenza precisa
  - l'algoritmo di Damas-Milner aggiunge le sostituzioni idempotenti (o cose equivalenti)
    - è stato dimostrato da Monsuez che si tratta dell'aggiunta di una specie di informazione relazionale, usata molto spesso nei domini astratti, per aumentare la precisione del calcolo astratto
  - la teoria dell'interpretazione atratta fornisce tecniche per il raffinamento sistematico di domini, che potrebbero essere molto utili per trasformare sistematicamente la proprietà di interesse in un buon dominio astratto
- come gestire l'approssimazione nei calcoli di punto fisso
  - l'interpretazione astratta ci dice quando possiamo tranquillamente calcolare punti fissi o quando dobbiamo usare operatori di widening