

- 1 Termination of queries
- 2 Termination for DEFINITE programs
- 3 Termination for NORMAL programs
  - Strong termination
  - LDNF-termination
- 4 Framework for AUTOMATED termination analysis
  - Introduction
  - LD-termination
  - Binary unfolding
  - Rounding up

## Acyclic program

### Level mapping

a function  $f : \text{ground literals} \rightarrow \mathcal{N}$   
with  $f(\text{not}(A)) = f(A)$   
maps every ground literal to *finite* value!

### Acyclic program

$P$  is **acyclic** if  
there exists a level mapping  $f$  such that  
for every  $A \leftarrow L_1, \dots, L_n \in \text{ground}(P)$ :  $\forall i : f(A) > f(L_i)$

As for definite programs (negation is ignored)

# Properties

## Termination of ground queries

SLDNF terminates on ground queries for acyclic programs

Proof

- infinite derivation instantiates to an infinite ground derivation
- contradicts acyclicity assumption

## Acyclicity of terminating programs

SLDNF can terminate for a ground query while the program is not acyclic (contrary to SLD/recurrency)

Proof:  $p(x) \leftarrow q, \text{not}(p(x))$  not acyclic

The query  $\leftarrow p(x)$  **finitely fails** because the computation rule is forced to select  $q$

$p(x) \leftarrow q, p(x)$  not acyclic

$r \leftarrow \text{not}(p(x))$

The query  $\leftarrow r$  **flounders**

# Properties

## Boundedness

Let  $f$  be a level mapping

A literal  $L$  is **bounded** wrt  $f$  if

$\{f(L\theta) \mid L\theta \text{ is ground}\}$  is bounded in  $\mathcal{N}$

## Termination of bounded queries

If  $P$  is acyclic wrt level mapping  $f$  and  $L$  is bounded wrt  $f$  then SLDNF terminates for all queries  $\leftarrow L$

## Boundedness does not imply success or failure

$evenlist([\ ])$   $\leftarrow$

$evenlist([X|Xs]) \leftarrow not(evenlist(Xs))$

Is acyclic with *length* as level mapping

$\leftarrow evenlist([X, Y, Z])$  is bounded but **flounders**

# Properties

## Other properties of acyclic programs

- An acyclic program is an inductive definition and is locally stratified

Because the level mapping divides  $ground(P)$  in strata

- $T_P$  has a unique fixpoint  $M_P$
- $M_P$  is the perfect model of  $T_P$
- $M_P$  is the unique Herbrand model of  $COMP(P)$
- $M_P$  is the unique model of  $IND(P)$
- $(M_P, B_P \setminus M_P)$  is the unique fixpoint of  $\Phi_P$
- $M_P$  is the unique well-founded model
- $M_P$  is the unique stable model

# Example: The Yale Turkey shooting problem

## The program

- (1)  $holds(alive, [ ])$  ←
- (2)  $holds(loaded, [load|S])$  ←
- (3)  $holds(dead, [shoot|S])$  ←  $holds(loaded, S)$
- (4)  $abnormal(alive, shoot, S)$  ←  $holds(loaded, S)$
- (5)  $holds(F, [E|S])$  ←  $not(abnormal(F, E, S)), holds(F, S)$

- (1) describes the initial state (true properties/**fluents**)
- (5) is the **frame axiom**, every fluent which is not abnormal for particular action  $E$  is preserved from previous state  $S$
- **wait** action preserves everything
- **load** action (2) results in loaded gun
- **shoot** action with **loaded** gun initiates **dead** (3) and terminates **alive** (4)

## Example: The Yale Turkey shooting problem

In the early eighties, logic was condemned

As FOL implications, there is a model where

- $abnormal(loaded, wait, [load])$  is true
- $holds(loaded, [wait, load])$  is false
- $holds(alive, [shoot, wait, load])$  is true
- $holds(dead, [shoot, wait, load])$  is false

## The logic program works

with SLDNF

- $\leftarrow \text{holds}(\text{dead}, [\text{shoot}, \text{wait}, \text{load}])$  succeeds
- $\leftarrow \text{holds}(\text{alive}, [\text{shoot}, \text{wait}, \text{load}])$  finitely fails

Because  $\text{abnormal}(\text{loaded}, \text{wait}, [\text{load}])$  **fails finitely**

Later it was realized that  $\text{abnormal}(\text{loaded}, \text{wait}, [\text{load}])$  is false when the program is considered

- as a locally stratified program
- as an inductive definition
- under the well-founded semantics

## Example: The Yale Turkey shooting problem

The program is acyclic

Level mapping  $||$ :

- $|holds(s, t)| = 2 \text{ length}(t)$
- $|abnormal(t1, t2, t3)| = 2 \text{ length}(t3) + 1$

with  $\theta$  a grounding substitution:

- (3):  $holds(dead, [shoot|S]) \leftarrow holds(loaded, S)$   
 $2 (\text{length}(S\theta) + 1) > 2 \text{ length}(S\theta)$
- (4)  $abnormal(alive, shoot, S) \leftarrow holds(loaded, S)$   
 $2 \text{ length}(S\theta) + 1 > 2 \text{ length}(S\theta)$
- (5)  $holds(F, [E|S]) \leftarrow not(abnormal(F, E, S)), holds(F, S)$   
atom 1:  $2(1 + \text{length}(S)\theta) > 2 \text{ length}(S\theta) + 1$   
atom 2:  $2(1 + \text{length}(S)\theta) > 2 \text{ length}(S\theta)$

# Acyclicity is not sufficient

Even when the computation rule has no choice

**Game** program

- $move(a_i, a_j) \leftarrow$   
*move*-facts: arcs in acyclic graph of possible moves
- $win(X) \leftarrow move(X, Y), not(win(Y))$
- $\leftarrow win(a_k)$  **strongly terminates** because safety of computation rule forces to select *move*-literal before the *win*-literal
- not acyclic E.g.  $win(a) \leftarrow move(a, a), not(win(a))$   
 $f(win(a)) > f(not(win(a)))$  is not possible

From now on consider the computation rule  
i.e. concentrate on LDNF-derivations

while the Game program is strong terminating, we immediately jump to left termination

## Acceptability for definite programs

$P$  is acceptable if there exists a **level mapping**  $f$  and a **model**  $I$  such that for each clause  $A \leftarrow B_1, \dots, B_k \in \text{ground}(P)$ :

**if**  $I \models B_1, \dots, B_{i-1}$  **then**  $\forall i : f(A) > f(B_i)$

Consider:  $A \leftarrow B_1, \dots, B_j, \text{not}(B_k), B_l, \dots \in \text{ground}(P)$

**if**  $I \models B_1, \dots, B_j, \text{not}(B_k)$  **then**  $f(A) > f(B_l)$ ?

The Herbrand base  $B_P$  is a model. This gives:

**if**  $\text{true} \wedge \dots \wedge \text{true} \wedge \text{false}$  **then**  $f(A) > f(B_l)$  or

**if**  $\text{false}$  **then**  $f(A) > f(B_l)$

Hence,  $B_l$  can be **unbounded** and non terminating

$A \leftarrow B_1 \dots, B_j, \text{not}(B_k), B_l, \dots \in \text{ground}(P)$

### One solution

- if  $I \models B_1, \dots, B_j$  then  $f(A) > f(B_k)$
- if  $I \models B_1, \dots, B_j$  then  $f(A) > f(B_l)$

This boils down to using the empty interpretation for the model of  $B_k$  (the negative literal is then always true)

This is fine when level mappings are based on size of data structures which is the case in automated termination analysers

Indeed, the successful execution of  $\text{not}(B_k)$  does not affect the size of data structures

But it is feasible to do better (when doing manual termination proofs)

## Requirements for the model $I$ to be used by acceptability

- If  $\text{not}(B_k)$  succeeds then  $B_k$  is in the finite failure set
- Then  $B_k$  does not belong to the greatest fixpoint of  $T_P$  and  $\text{not}(B_k)$  is true in models smaller or equal to the greatest fixpoint:  $I|_{B_k} \subseteq \text{gfp}(T_P)|_{B_k}$
- Positive atoms that succeed are part of  $\text{lfp}(T_P)$ , they are true in any model larger or equal to  $\text{lfp}(T_P)$ :  
 $\forall i : 1 \leq i \leq j : I|_{B_i} \supseteq \text{lfp}(T_P)|_{B_i}$
- If an atom has both positive and negative occurrences: One could use one model for the positive and one model for the negative occurrences.
- Practical: choose  $I|_{B_k} = \emptyset$  or, with  $P_{B_k}$  the part of the program defining predicate  $B_k$ , prove that  $I|_{P_{B_k}}$  is a fixpoint of  $T_{P_{B_k}}$  hence a model of the completion and thus also for the positive atoms in  $P_{B_k}$   
 For each of the remaining predicates  $B_i$ : prove that  $I|_{P_{B_i}}$  is a model of  $P_{B_i}$ , the part of the program defining the predicate.

## Game is acceptable

Let  $G$  be the (acyclic) graph of moves

- $f(a) =$  if for no  $b : (a, b) \in G$  then 0  
else  $1 + \max(\{f(b) \mid (a, b) \in G\})$

$f(a)$ : length of the longest sequence of moves starting in  $a$

- $g(a) =$  if for no  $b : (a, b) \in G$  then 0  
else  $1 - \min(\{g(b) \mid (a, b) \in G\})$

In a leaf and any other losing position: 0.

In a winning position: 1

- $I = \{\text{move}(a, b) \mid (a, b) \in G\} \cup \{\text{win}(a) \mid g(a) = 1\}$

- level mappings  $||$ :

$$|\text{move}(a, b)| = f(a)$$

$$|\text{win}(a)| = f(a) + 1$$

## Game is acceptable (cont)

First we prove that  $I$  is a fixpoint (a model of the completion) of  $T_{win}$ , the part of the program defining  $win$  which is the whole program

$$T_{Game}(I) \subseteq I$$

- $move(a, b) \in I$  iff  
 $(a, b) \in G$  iff  
 a fact  $move(a, b) \leftarrow \in Game$  iff  
 $move(a, b) \in T_{Game}(I)$
- assume  $win(a) \in T_{Game}(I)$   
 thus  $I \models move(a, b), not(win(b))$   
 $move(a, b) \in I$  thus  $(a, b) \in G$   
 $not(win(b))$  is true in  $I$  thus  $g(b) = 0$   
 thus  $g(a) = 1 - \min(\{0, \dots\}) = 1$   
 thus  $I \models win(a)$

## Game is acceptable (cont)

$I$  is a fixpoint (cont)

$$I \subseteq T_{Game}(I)$$

- $move(a, b) \in I$  (see above)  
iff  $move(a, b) \in T_{Game}(I)$
- assume  $win(a) \in I$  thus  $g(a) = 1$   
thus  $\exists b : (a, b) \in G$  and  $g(b) = 0$   
thus  $move(a, b) \in I$  and  $win(b) \notin I$   
for this  $b : I \models move(a, b), not(win(b))$   
thus  $win(a) \in T_{Game}(I)$

Hence  $I$  is a fixpoint of the  $T_{Game}$  operator and a model of the completion

## Game is acceptable (cont)

Consider  $win(a) \leftarrow move(a, b), not(win(b)) \in ground(Game)$

- $|win(a)| \stackrel{?}{>} |move(a, b)|$   
yes because  $f(a) + 1 > f(a)$
- if  $I \models move(a, b)$   
then  $|win(a)| \stackrel{?}{>} |not(win(b))|$   
 $f(a) + 1 \stackrel{?}{>} f(b) + 1$   
move  $(a, b) \in G$  thus  
length longest path from  $a$  is greater than length longest path  
from  $b$   
thus  $f(a) > f(b)$

Hence **Game is acceptable**

## Note

We did not need the part of the model about *win*, we only used the part about *move*. It was an overkill to choose a model of the completion; a model for *move* was sufficient. So this was a not so good example

## Exercise

3. Consider the following program:

$\text{even}(0) \leftarrow.$

$\text{even}(X) \leftarrow \text{pred}(X, Y), \text{not}(\text{even}(Y)).$

$\text{pred}(s(X), X) \leftarrow.$

- Is this an acyclic program? Explain.
- Find a level mapping and a model such that the program is acceptable.
- What follows for the termination of the query  $\leftarrow \text{even}(s(s(s(s(0)))))$ . ?
- And for termination of the query  $\leftarrow \text{even}(s(s(X)))$ . ?

## Limitations of automation

- There is no decision procedure
- Hence, the best is to try to find **sufficient** conditions for termination (or for non-termination)
- One cannot prove termination of all programs e.g. *Game* is very hard to automate but is a functor free program: terminating procedures exist
- Search for a level mapping that allows one to prove termination?

## Level mappings?

Level mappings as seen so far are **unnatural** because

- They assign a level mapping to **all** predicates
- Level mappings achieve two unrelated things:
  - if  $p(\bar{t})$  **recursively** calls  $p(\bar{t}')$  then the SLD-tree for  $p(\bar{t}')$  is **strictly smaller** than the SLD-tree for  $p(\bar{t})$
  - if  $p(\bar{t})$  calls  $q(\bar{t}')$  then the SLD-tree for  $q(\bar{t}')$  is **bounded** (finite)

Also the second is essential!

$p([\ ])$  ←

$p([H|T])$  ←  $append(X, Y, Z), p(T)$

is nonterminating for ground calls,

e.g. ←  $p([a, b])$

## General framework

- Focus on **loops** of the program  
Each time control reaches the same program point, there should be a decrease in the measure (element of a well-founded set) that captures the size of the program state.  
In LP: program points are calls to predicates, loops are calls to the same predicate
- Compute **Calls** (a safe approximation) of all atoms (literals) which are selected for execution (for the queries of interest)
- Focus on **recursion** of the program  
For each call  $p(\bar{t})$  in *Calls*  
When the execution calls  $p(\bar{s})$   
Then, show  $p(\bar{s})$  is smaller than  $p(\bar{t})$
- Use **Level Mappings** to measure size of calls  
No interaction between level mappings for different predicates.

# Collecting calls

An operator  $T_P^C$

Mapping sets of (nonground) atoms to sets of atoms

$$T_P^C(S) = S \cup \{b_i\theta\sigma_1 \dots \sigma_{i-1} \mid$$

- $h \leftarrow b_1, \dots, b_{i-1}, b_i, \dots \in P$
- $a \in S$
- $\theta = mgu(a, h)$
- $\sigma_1$  is a cas of  $b_1\theta$
- $\sigma_2$  is a cas of  $b_2\theta\sigma_1$
- ...
- $\sigma_{i-1}$  is a cas of  $b_{i-1}\theta\sigma_1 \dots \sigma_{i-2}$

# Collecting calls

## $Calls(C, P)$

- $T_P^C$  is monotone
- $T_P^C$  is continuous
- $T_P^C$  has a least fixpoint:

$$Calls(C, P) = \text{Ifp}(T_P^C(C))$$

It is the set of all atoms that can be selected in an LD-derivation of a query  $\leftarrow a$  for  $a \in C$

- A program  $P$  is left terminating wrt a set of atomic calls  $C$  iff it is left terminating for  $Calls(C, P)$

## Left Recursive Resultant

- Built a **partial** LD-tree for  $p(\bar{t})$ : terminate a branch when the selected atom is  $p(\bar{s})$
- For each branch that represents a partial derivation:  
Extract the resultant:  $p(\bar{t})\theta \leftarrow p(\bar{s}), \dots$   
 $\theta$  is the substitution accumulated along the branch  
(One can drop the “...” part)

## Level mapping revisited

- A mapping  $f$  from atoms to natural numbers
- $f(A) = \min(\{f(A\theta) \mid \theta \text{ is a grounding substitution}\})$
- A level mapping is a **terminating condition** for a left recursive resultant  $p(\bar{t})\theta \leftarrow p(\bar{s})$  iff  $f(p(\bar{t})\theta) > f(p(\bar{s}))$
- A level mapping  $f$  for  $p(\bar{t})$  is **rigid** if  $f(p(\bar{t})) = f(p(\bar{t})\theta)$  for all  $\theta$

## Global Termination Condition

### acceptability

$P$  is *acceptable* wrt set of calls  $C$  if there is a level mapping  $f$  for each recursive predicate such that

- for each call  $p(\bar{t})$  in  $C$
- for each left recursive resultant  $p(\bar{t})\theta \leftarrow p(\bar{s})$  of  $p(\bar{t})$
- $f(p(\bar{t})) > f(p(\bar{s}))$

When  $f$  is rigid for each call in  $C$ :

- $f$  is a terminating condition for each left recursive resultant  $p(\bar{t})\theta \leftarrow p(\bar{s})$   
i.e.:  $f(p(\bar{t})\theta) > f(p(\bar{s}))$

# Summary

## What we did

A reformulation of acceptability that separated concerns:

- Computation of the set  $C$  of calls allows us to focus on each predicate separately
- Level mapping maps (SLD trees of) calls to elements of a well-founded set
- Condition on left recursive resultant ensures that the SLD-tree of recursive call is definitely smaller than the SLD-tree of original call
- Effect: easier to find appropriate level mappings
- Note that  $C$  and the set of recursive resultants can be infinite

# Example

$$\begin{aligned}
 & perm([], []) \leftarrow \\
 & perm([X|Xs], [Y|Ys]) \leftarrow delete(Y, [X|Xs], Zs), \\
 & \qquad \qquad \qquad perm(Zs, Ys)
 \end{aligned}$$

$$\begin{aligned}
 & delete(X, [X|Xs], Xs) \leftarrow \\
 & delete(Y, [X|Xs], [X|Zs]) \leftarrow delete(Y, Xs, Zs)
 \end{aligned}$$

$$C = \{ perm(x, y) \mid x \text{ is a } nil\text{-terminated list of free variables} \\
 \qquad \qquad \qquad y \text{ is a free variable} \}$$

$P$  is not recurrent (select  $perm/2$  before  $delete/3$ )

$P$  is acceptable, i.e. left terminating

## Example (cont)

### call set

- $perm(nillist, var)$  calls  $delete(var, nillist, var)$
- $delete(var, nillist, var)$  calls  $delete(var, nillist, var)$
- $delete(var, nillist, var)$  succeeds with  $delete(var, nillist, nillist)$
- $perm(nillist, var)$  calls  $perm(nillist, var)$

$$T_P^C(C) = \text{lfp}(T_P^C(C)) =$$

$$\left\{ \begin{array}{l} perm(x, y) \\ \left. \begin{array}{l} x \text{ is } nil\text{-terminated list of} \\ \text{free variables} \\ y \text{ is free variable} \end{array} \right\} \cup \\ \left\{ \begin{array}{l} delete(x, y, z) \\ \left. \begin{array}{l} y \text{ is } nil\text{-terminated list} \\ \text{of free variables} \\ x, z \text{ are free variables} \end{array} \right\} \end{array} \right.$$

## calls $delete(var, nilist, var)$

- Levelmapping  $f(delete(x, y, z)) = listlength(y)$
- Is rigid for calls  $delete(var, nilist, var)$
- Its left recursive resultants are of the form:
- $delete(Y, [V_1, \dots, V_n], [V_1|Zs]) \leftarrow delete(Y, [V_2, \dots, V_n], Zs)$
- $f(delete(Y, [V_1, \dots, V_n], [V_1|Zs])) = n > n - 1 = f(delete(Y, [V_2, \dots, V_n], Zs))$
- Hence  $f$  is a terminating condition for the left recursive resultants

## calls $perm(nillist, var)$

- Levelmapping  $f(perm(x, y)) = listlength(x)$
- Is rigid for calls  $perm(nillist, var)$
- Answers for  $delete(X, [V_1, \dots, V_n], Y)$  are of the form  $delete(X, [V_1, \dots, V_n], [W_1, \dots, W_{n-1}])$
- Hence left recursive resultants of  $perm([V_1, \dots, V_n], Y)$  are of the form  $perm([W_1, \dots, W_{n-1}], Zs)$
- $f(perm([V_1, \dots, V_n], Y)) = n > n - 1 = f(perm([W_1, \dots, W_{n-1}], Zs))$
- Hence  $f$  is a terminating condition for the left recursive resultants

The program is acceptable for the given set of calls

## Binary Unfolding of a program

A binary unfolding operator  $T_P^b$

Maps binary clauses to binary clauses

$T_P^b(B) = \{(h \leftarrow b)\theta \text{ such that: } h \leftarrow b_1, \dots, b_m \text{ is a clause in } P$

- for some  $i \in [1..m]$ :  $\{h_1 \leftarrow true, \dots, h_{i-1} \leftarrow true\} \subseteq B$   
 $\theta = mgu((h_1, \dots, h_{i-1}), (b_1, \dots, b_{i-1}))$   
 $b = b_i$  OR
- for some  $i \in [1..m]$ :  
 $\{h_1 \leftarrow true, \dots, h_{i-1} \leftarrow true, h_i \leftarrow c\} \subseteq B$  ( $b_i \neq true$ )  
 $\theta = mgu((h_1, \dots, h_i), (b_1, \dots, b_i))$   
 $b = c$  OR
- $\{h_1 \leftarrow true, \dots, h_m \leftarrow true\} \subseteq B$   
 $\theta = mgu((h_1, \dots, h_m), (b_1, \dots, b_m))$   
 $b = true$  }

## Binary Unfolding of a program

- The binary unfolding is the least fixpoint of  $T_P^b$ , i.e.:  
$$\text{Bin\_unf}(P) = \text{lfp}(T_P^b)$$
- It is **goal independent**, i.e., independent from a particular query
- It is **closed under unfolding**: if  $p(x) \leftarrow p(y)$  and  $p(y') \leftarrow p(z) \in \text{Bin\_unf}(P)$  and  $\text{mgu}(p(y), p(y')) = \theta$  then  $(p(x) \leftarrow p(z))\theta \in \text{Bin\_unf}(P)$
- It allows to compute all **calls** selected during an LD-derivation of an atomic call  $\leftarrow c$  or a set of atomic calls  $C$   
$$\text{Calls}(P, C) = C \cup \{b\theta \mid c \in C, h \leftarrow b \in \text{Bin\_unf}(P), \theta = \text{mgu}(c, h)\}$$
- It allows to compute all **answers** to  $\leftarrow c$   
$$\{\theta \mid h \leftarrow \text{true} \in \text{Bin\_unf}(P), \theta = \text{mgu}(c, h)\}$$

# Binary Unfolding of a program

## Calls\_to relation

**Calls\_to relation** of a call  $c$ :

- If the successively selected calls in an LD-derivation of  $\leftarrow c$  are  $c = a_0, a_1, a_2, \dots, a_n$
- Then, for every  $0 \leq i < j \leq n$ :  $(a_i, a_j) \in \text{Calls\_to}(P, \{c\})$
- It can be derived from the binary unfolding:  
 $(a, b)$  is in the Calls\_to relation for a call  $\leftarrow c$  iff  $a$  is selected in the LD-derivation for  $\leftarrow c$  and  $b$  is selected in the LD-derivation for  $\leftarrow a$
- If  $a \in \text{Calls}(P, \{c\})$  and  $b \in \text{Calls}(P, \{a\})$  then  $(a, b) \in \text{Calls\_to}(P, \{c\})$
- $\text{Calls\_to}(P, \{c\}) = \{(a, b\theta) \mid a \in \text{Calls}(P, \{c\}), h \leftarrow b \in \text{Bin\_unf}(P), \theta = \text{mgu}(a, h)\}$

## Binary Unfolding of a program

- An infinite sequence of selected calls in a derivation of  $\leftarrow c$  exists **iff** one can construct an infinite sequence  $(a_0, a_1), (a_1, a_2), \dots$  from pairs in the *Calls\_to* relation of  $\leftarrow c$
- The binary unfolding preserves **termination**: A query  $\leftarrow c$  terminates for  $P$  **iff** it terminates for  $Bin\_unf(P)$
- A nonterminating sequence of selected calls  $a_0 a_1, \dots$  exists **iff** a nonterminating subsequence  $p(\bar{t}_0), p(\bar{t}_1), \dots$  exists for some predicate  $p$
- The latter can be constructed from the pairs  $(p(\dots), p(\dots))$  in the *Calls\_to* relation (they are the **loops**)
- These loops originate from binary recursive clauses in  $Bin\_unf(P)$
- Hence, to prove termination: sufficient to study binary recursive clauses

## Binary Unfolding of a program

### The global approach

- A level mapping  $f$  for each predicate  $p$  such that for each binary recursive clause  $p(\bar{t}) \leftarrow p(\bar{s})$
- If  $p(\bar{c}) \in \text{Calls}(P)$  and  $\theta = \text{mgu}(\bar{c}, \bar{t})$  (hence  $(p(\bar{c}), p(\bar{s})\theta)$  is a **loop**)
- then  $f(p(\bar{c})) > f(p(\bar{s})\theta)$
- **If  $f$  is rigid** then  $f(p(\bar{c})) = f(p(\bar{c})\theta) = f(p(\bar{t})\theta)$
- Hence  $f(p(\bar{t})\theta) > f(p(\bar{s})\theta)$
- Or the level mapping is a **terminating condition** for the binary clauses  $p(\bar{t}) \leftarrow p(\bar{s})$

# Binary Unfolding of a program

## The local approach

- One can use a **different** level mapping for each binary clause as long as the set of level mappings is finite
- Based on Ramsey's Theorem (1930)

# The local approach

## Ramsey's Theorem

- Let  $C = \{(a, b) \mid a, b \in N, a < b\}$  hence infinite  
( $a$  and  $b$  are the indices of the sequence of the selected calls for some predicate  $p$  in the LD-derivation of a query  $\leftarrow c$  such that  $(p_a, p_b) \in \text{Calls\_to}(P, \{c\})$ )
- Let  $L$  be a finite set of colours (level mappings)
- Let  $F$  be a function associating a colour (level mapping) with each pair  $(a, b)$
- Then there is a colour (level mapping) and an infinite subset  $X$  of  $N$  (an infinite subsequence of selected atoms of the same predicate) such that for each  $a < b$  with  $a, b \in X$   $(a, b)$  has the same colour (level mapping)

## The local approach

### Corollary

- A finite set of level mappings  $L$
- For every loop in  $Calls\_to(P, \{c\})$  there is a level mapping  $f \in L$  such that  $f$  is a terminating condition for it
- Then every LD-derivation for  $\leftarrow c$  is terminating

### Indeed

- Assume an infinite sequence of selected calls exists, then, according to Ramsey's theorem, there is an infinite sequence  $p_0, p_1, \dots$  such that, for every loop  $(p_i, p_j)$  ( $i < j$ ), the same level mapping  $f$  is assigned.
- But then  $f(p_0) > f(p_1) > f(p_2) > \dots$  because  $f$  is a terminating condition which is impossible because the range of  $f$  is a well-founded set.

## The local approach

### Example: Ackermann function

$ack(0, N, s(N))$ .

$ack(s(M), 0, R) : -ack(M, s(0), R)$ .

$ack(s(M), s(N), R) : -ack(s(M), N, R1), ack(M, R1, R)$ .

- query:  $\leftarrow ack(m, n, R)$ . with  $m, n$  of the form  $s(s(\dots(0)\dots))$
- All calls of the form  $ack(m, n, R)$
- Level mappings:  $f1(ack(m, n, R)) = m$  and  $f2(ack(m, n, R)) = n$ .
- Basic loops:  
 $(ack(s(m), ?, R), ack(m, ?, R))$ : decrease by  $f1$   
 $(ack(m, s(n), R), ack(m, n, R))$ : decrease by  $f2$
- Global approach needs a more complex measure  
 $f(ackermann(m, n, R)) = (m, n)$  and lexicographic order

- Not as simple as checking clauses (in the manual methods) but **natural level mappings** for programs manipulating data structures not for tricky programs (cnfr Game)
- **natural**: size of input data structure. E.g. number of elements in a list, number of internal nodes in a tree, number of functors in a term, ...
- How to compute? (and to approximate, it can be infinite)  
By **abstract interpretation**
- How to determine rigid part of the calls?  
By **abstract interpretation**
- How to approximate computed answer substitutions?  
needed for acceptability or for constructing the binary unfolding  
By **abstract interpretation**: as relations between size of arguments e.g.  
 $perm(x, y) : |x| = |y|$   $delete(x, y, z) : |z| + 1 = |y|$   $sort(x, y) : |x| = |y|$

Exercise 4. Consider the following program:

$qs([],[]) \leftarrow .$

$qs([X|L],S) \leftarrow part(X,L,L1,L2), qs(L1,S1), qs(L2,S2), append(S1,[X|S2],S).$

$part(X,[],[],[]) \leftarrow .$

$part(X,[Y|L],[Y|L1],L2) \leftarrow X \geq Y, part(X,L,L1,L2).$

$part(X,[Y|L],L1,[Y|L2]) \leftarrow X < Y, part(X,L,L1,L2).$

$append([],L,L) \leftarrow .$

$append([X|U],V,[X|W]) \leftarrow append(U,V,W).$

Consider the set of calls  $S = \{qs(X,Y) \mid X \text{ is a list of integers and } Y \text{ is free}\}.$

- Is the program strongly terminating wrt.  $S$ ?
- Follow the framework for automated termination analysis to show that the program is left-terminating wrt.  $S$ .