# Use of SMT Solvers in Verification

Thomas Wies

New York University

# Overview

**Part 1**

**Use of Craig Interpolants in Fault Localization**

**Part 2**

**Computing Craig Interpolants**

**Part 1**

**Use of Craig Interpolants in Fault Localization:**

Error Invariants [FM’12]

joint work with

Evren Ermis (Freiburg University, Germany)

Martin Schäf (United Nations University, IIST, Macau)

# Faulty Shell Sort

Program

- takes a sequence of integers as input
- returns the sorted sequence.

On the input sequence 11, 14 the program returns 0, 11 instead of 11,14.

```c
#include<stdio.h>
#include<stdlib.h>

static void shell_sort(int a[], int size)
 {
   int i, j;
   int h = 1;
   do {
        h = h * 3 + 1;
   } while (h <= size);
   do {
        h /= 3:
        for (i = h; i < size; i++) {
           int v = a[i];
           for (j = i; j >= h && a[j - h] > v; j -= h)
              a[j] = a[j-h];
           if (i != j)
              a[j] = v;
         }
   } while (h != 1);
 }
int main(int argc, char *argv[])
 {
    int i = 0;
    int *a = NULL;

    a = (int *)malloc((argc-1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
       a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc);

    for (i = 0; i < argc - 1; i++)
       printf("%d", a[i]);
    printf("\n");

    free(a);
    return 0;
 }
```

# Error Trace

```
int i,j, a[];
int size=3;
int h=1;
h = h*3+1;
assume !(h<=size);
h/=3;
i=h;
assume (i<size);
v=a[i];
j=i;
assume !(j>=h && a[j-h]>v);
i++;
assume (i<size);
v=a[i];
j=i;
assume (j>=h && a[j-h]>v);
a[j]=a[j-h];
j-=h;
assume (j>=h && a[j-h]>v);
a[j]=a[j-h];
j-=h;
assume !(j>=h && a[j-h]>v);
assume (i!=j);
a[j]=v;
i++;
assume !(i<size);
assume (h==1);
assert a[0] == 11 && a[1] == 14;
```

# Error Trace

Input Values | Control-Flow Path | Expected Outcome

Can be obtained, e.g.,

- from a static analysis tool;
- from a failing test case;
- during debugging.

# The Fault Localization Problem

Error traces

- can become very long (thousands of statements);
- contains many statements and program variables that are irrelevant for understanding the error;
- provide no explicit information about the program state.

Fault Localization:

- Identify the relevant statements and variables.
- Provide an explanation for the error that incorporates the state of the program.

# Error Trace

```
0 int i,j, a[];
1 int size=3;
2 int h=1;
3 h = h*3+1;
4 assume !(h<=size);
5 h/=3;
6 i=h;
7 assume (i<size);
8 v=a[i];
9 j=i;
10 assume !(j>=h && a[j-h]>v);
11 i++;
12 assume (i<size);
13 v=a[i];
14 j=i;
15 assume (j>=h && a[j-h]>v);
16 a[j]=a[j-h];
17 j-=h;
18 assume (j>=h && a[j-h]>v);
19 a[j]=a[j-h];
20 j-=h;
21 assume !(j>=h && a[j-h]>v);
22 assume (i!=j);
23 a[j]=v;
24 i++;
25 assume !(i<size);
26 assume (h==1);
27 assert a[0] == 11 && a[1] == 14;
```

# Error Invariants

An error invariant $I$ for a position $i$ in an error trace $\tau$ is a formula over program variables s.t.

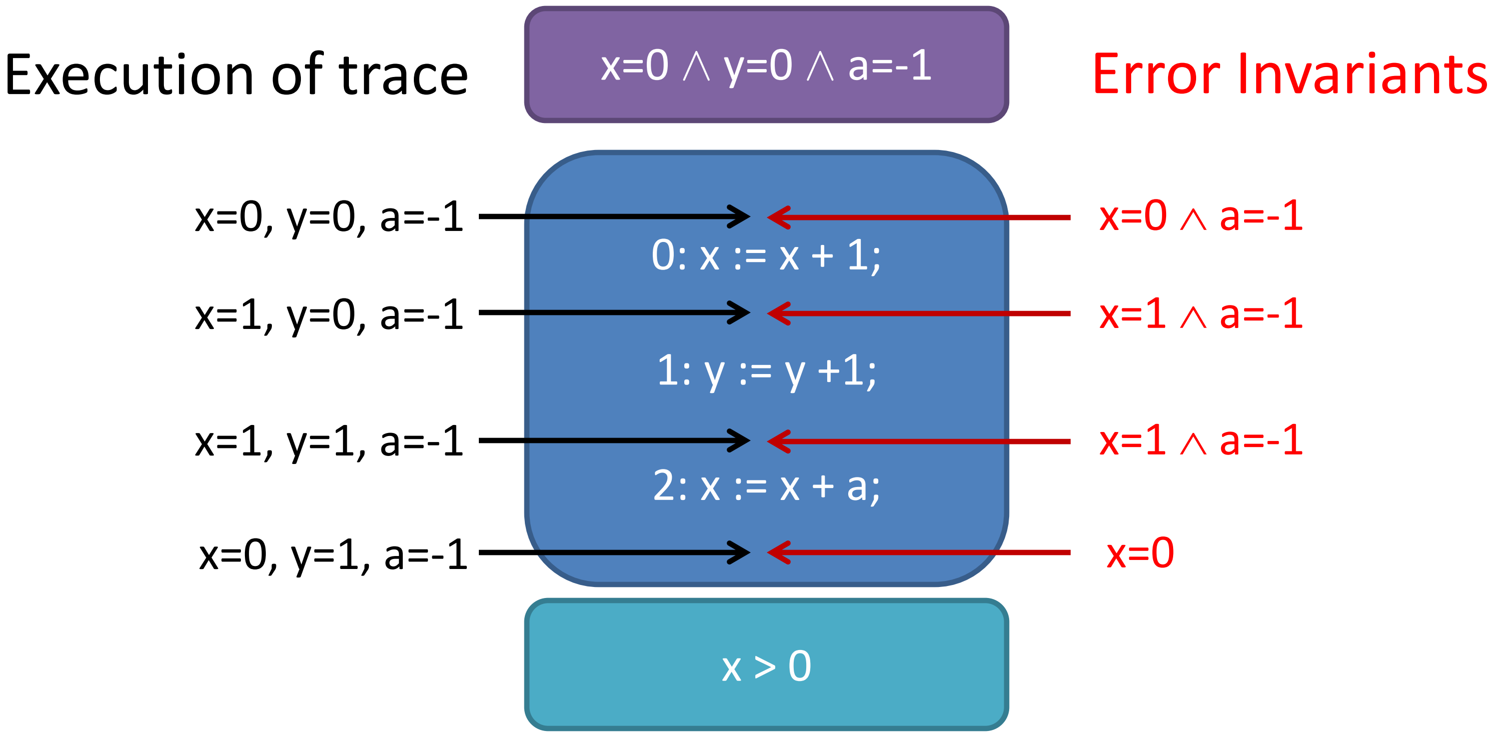
– all states reachable by executing the prefix of $\tau$ up to position $i$ satisfy $I$

– all executions of the suffix of $\tau$ that start from $i$ in a state that satisfies $I$, still lead to the error.

$I$ is called an inductive error invariant for positions $i < j$ if $I$ is an error invariant for both $i$ and $j$.

# Error Invariants

## Example




Information provided by the error invariants

- Statement y := y + 1 is irrelevant
- Variable y is irrelevant
- Variable a is irrelevant after position 2

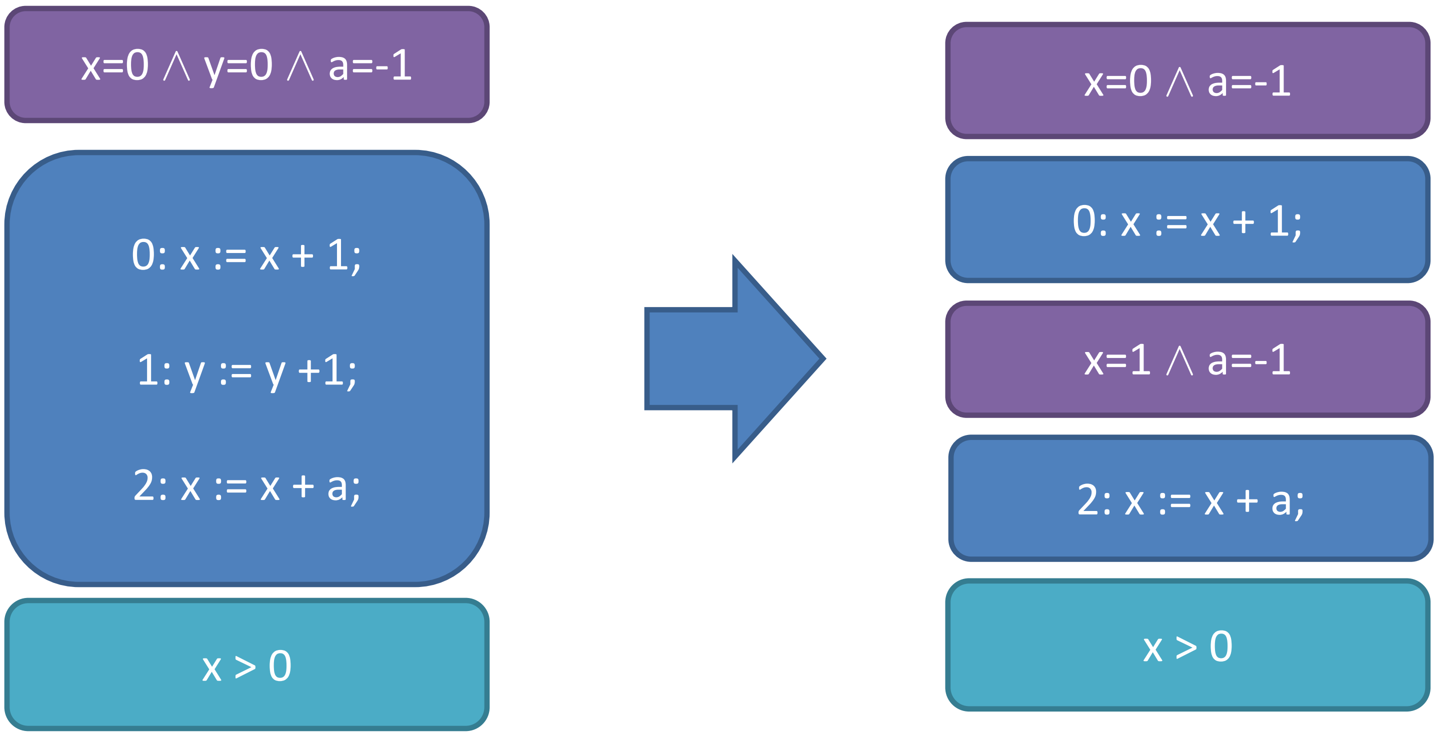
# Abstract Error Trace

Abstract error trace consists only of

- relevant statements and
- error invariants that hold before and after these statements.

# Abstract Error Trace

## Example



# Abstract Error Trace for Faulty Shell Sort

```
0 int i,j, a[];
1 int size=3;
2 int h=1;
3 h = h*3+1;
4 assume !(h<=size);
5 h/=3;
6 i=h;
7 assume (i<size);
8 v=a[i];
9 j=i;
10 assume !(j>=h && a[j-h]>v);
11 i++;
12 assume (i<size);
13 v=a[i];
14 j=i;
15 assume (j>=h && a[j-h]>v);
16 a[j]=a[j-h];
17 j-=h;
18 assume (j>=h && a[j-h]>v);
19 a[j]=a[j-h];
20 j-=h;
21 assume !(j>=h && a[j-h]>v);
22 assume (i!=j);
23 a[j]=v;
24 i++;
25 assume !(i<size);
26 assume (h==1);
27 assert a[0] == 11 && a[1] == 14;
```

$a[2]=0$

6: i := h;

$a[2]=0 \wedge h=1 \wedge i=h$

11: i := i+1;

$a[2]=0 \wedge h=1 \wedge i=2$

13: v := a[i];

$h=1 \wedge i=2 \wedge v=0 \wedge h \leq j \wedge j \geq 1$

20: j := j - h;

$h=1 \wedge i=2 \wedge v=0 \wedge j=0$

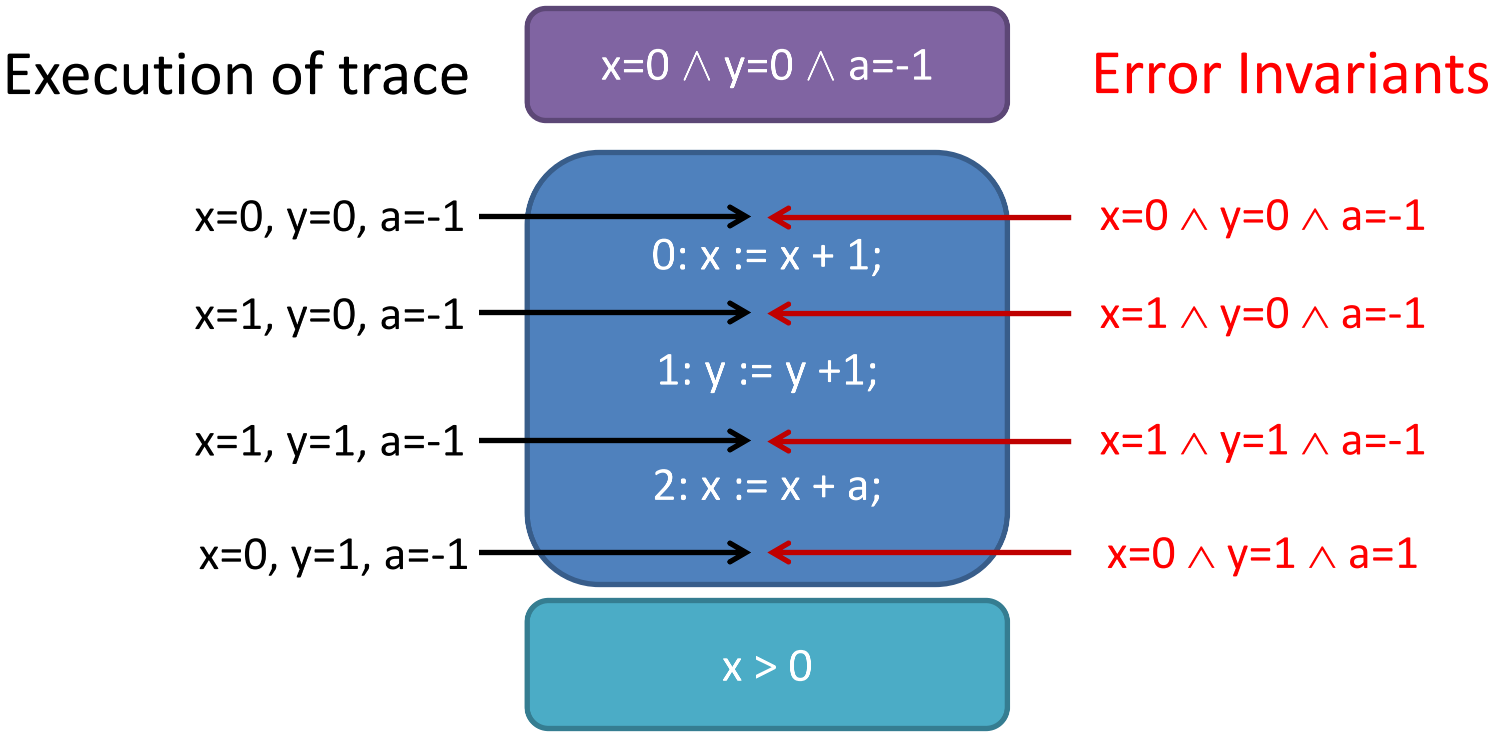
23: a[j] := v;

$a[0]=0$

$x > 0$

# How can we compute error invariants?

# Error invariants are not unique




We are interested in **inductive** error invariants!

# Checking Error Invariants

Input Values | Control-Flow Path | Expected Outcome

error trace formula

$\nvDash$ Precondition $\wedge$ Path Formula $st_0 \wedge \ldots \wedge st_i \wedge \ldots \wedge st_n$ $\wedge$ Postcondition

# Error Trace Formula

## Example

$x=0 \wedge y=0 \wedge a=-1$

0: x := x + 1;

1: y := y +1;

2: x := x + a;

$x > 0$

$$x_0=0 \wedge y_0=0 \wedge a_0=-1$$

$$\wedge$$

$$x_1 = x_0 + 1 \wedge$$

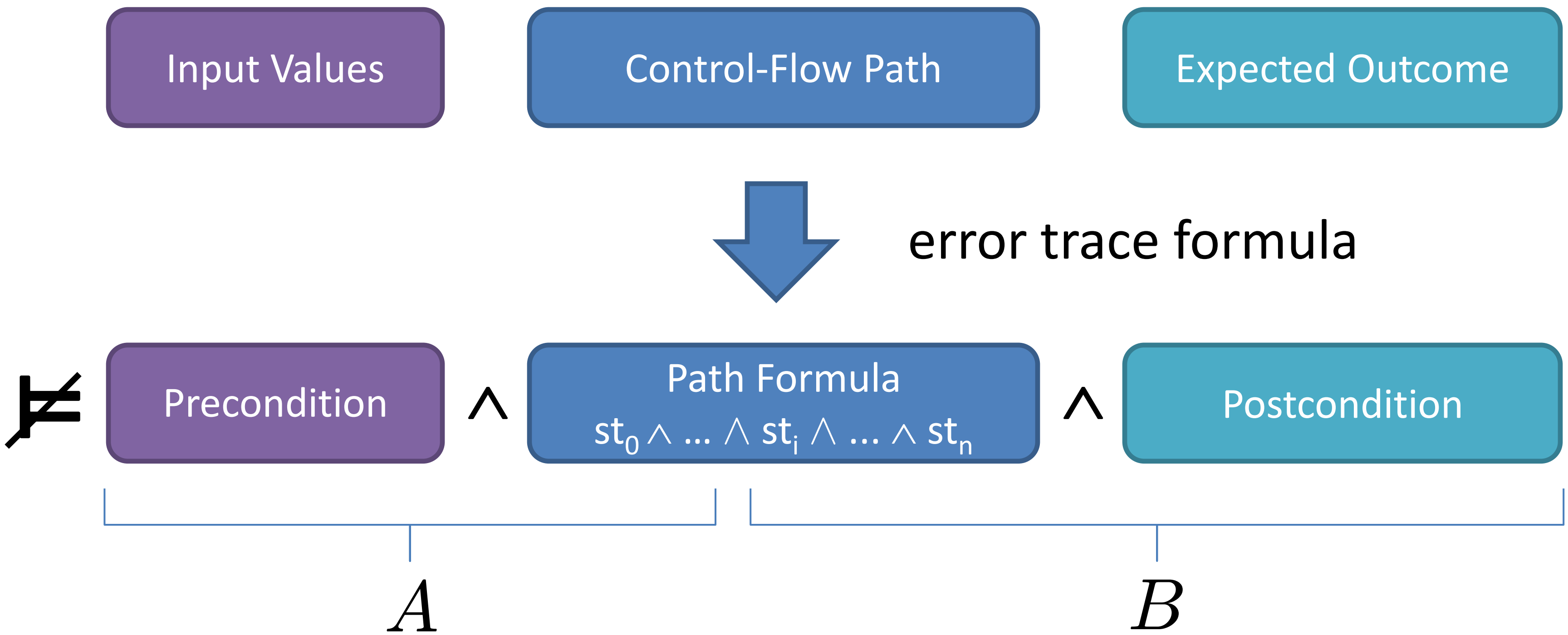
$$y_1 = y_0 + 1 \wedge$$

$$x_2 = x_1 + a_0$$

$$\wedge$$

$$x > 0$$

# Checking Error Invariants




$I$ is an error invariant for position $i$ iff

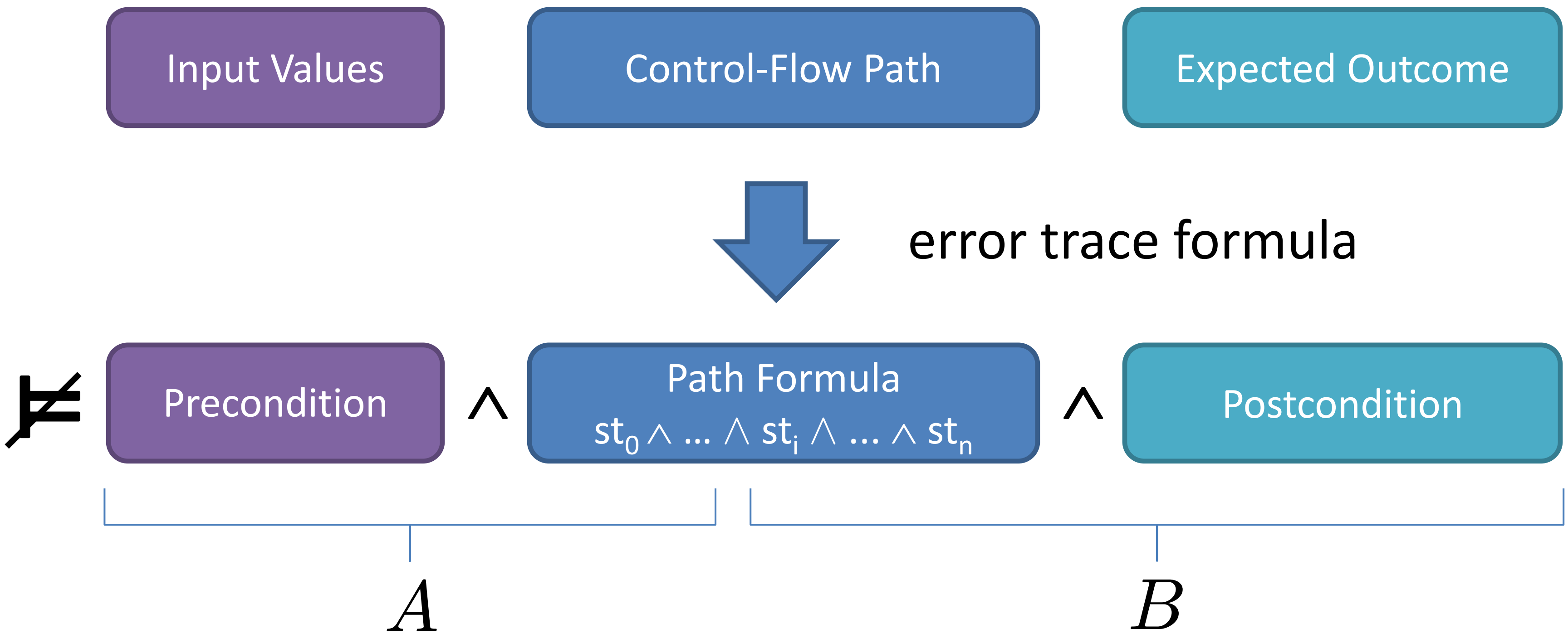
$$A \models I \quad \text{and} \quad I \wedge B \models \bot$$

# Craig Interpolants

Given formulas $A$, $B$ whose conjunction is unsatisfiable, a Craig interpolant for ($A$, $B$) is a formula $I$ such that

- $A \models I$
- $I \land B \models \bot$
- fv($I$) $\subseteq$ fv($A$) $\land$ fv($B$)

# Craig Interpolants are Error Invariants




Craig interpolant for $A \wedge B$ is an error invariant for position $i$

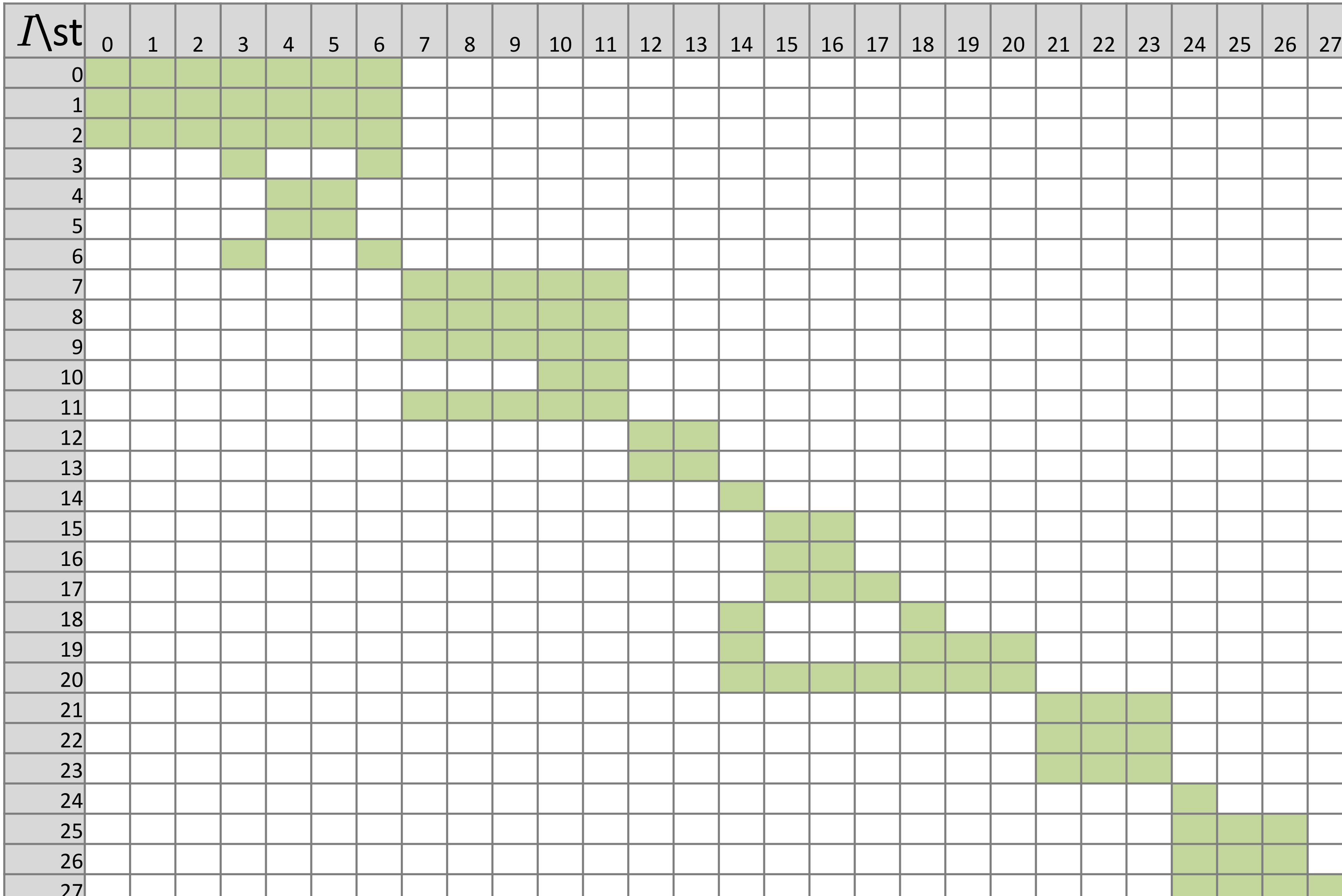
$\Rightarrow$ use Craig interpolation to compute candidates for inductive error invariants.

# Computing Abstract Error Traces

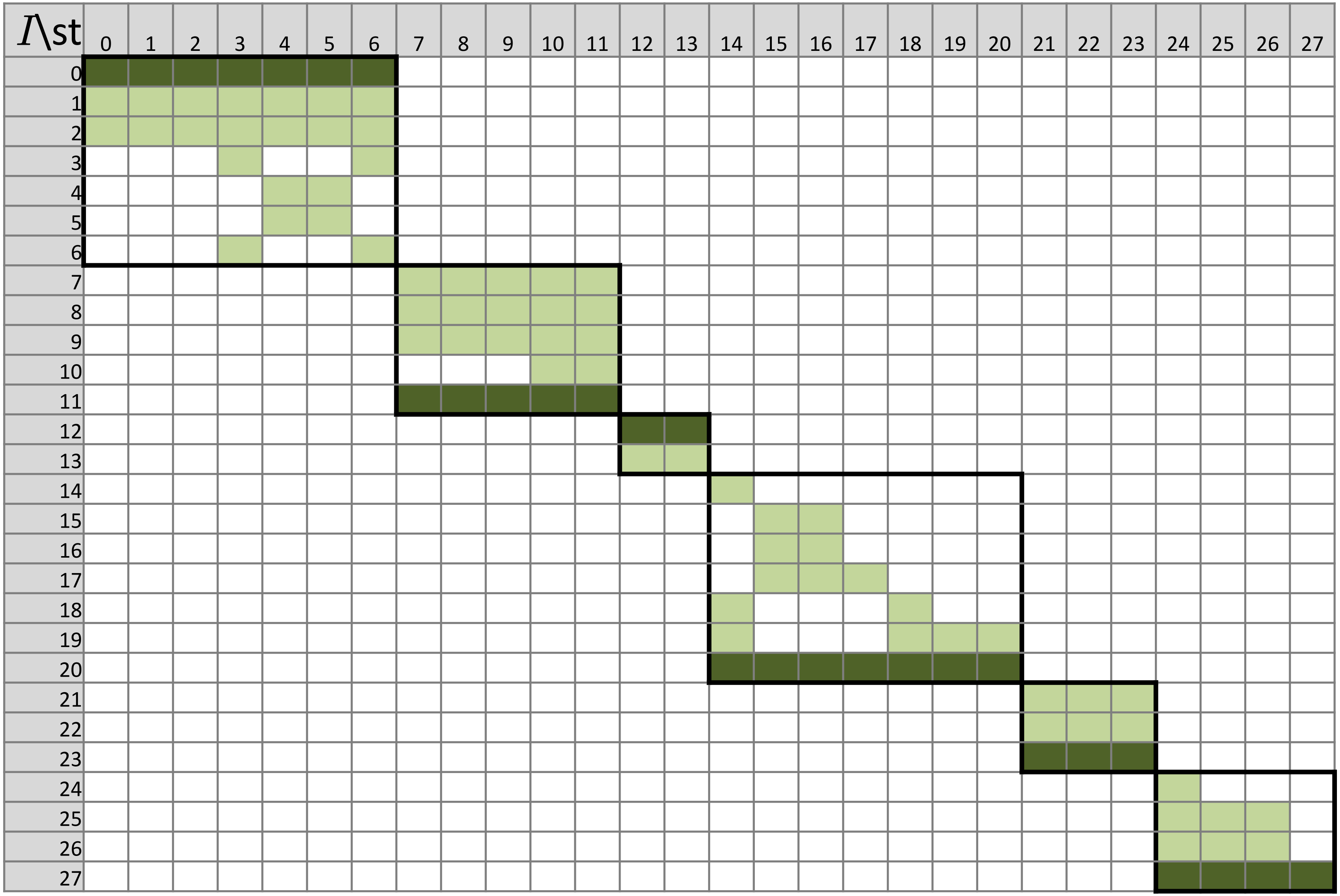
Basic Algorithm:

1. Compute the error trace formula from the error trace.
2. Compute a Craig interpolant $I_i$ for each position $i$ in the error trace.
3. Compute the error invariant matrix:
   – for each $I_i$ and $j$, check whether $I_i$ is an error invariant for $j$.
4. Choose minimal covering of error trace with inductive error invariants.
5. Output abstract error trace.

# Error Invariant Matrix for Shell Sort



# Error Invariant Matrix for Shell Sort



# Abstract Error Trace for Faulty Shell Sort

```
0 int i,j, a[];
1 int size=3;
2 int h=1;
3 h = h*3+1;
4 assume !(h<=size);
5 h/=3;
6 i=h;
7 assume (i<size);
8 v=a[i];
9 j=i;
10 assume !(j>=h && a[j-h]>v);
11 i++;
12 assume (i<size);
13 v=a[i];
14 j=i;
15 assume (j>=h && a[j-h]>v);
16 a[j]=a[j-h];
17 j-=h;
18 assume (j>=h && a[j-h]>v);
19 a[j]=a[j-h];
20 j-=h;
21 assume !(j>=h && a[j-h]>v);
22 assume (i!=j);
23 a[j]=v;
24 i++;
25 assume !(i<size);
26 assume (h==1);
27 assert a[0] == 11 && a[1] == 14;
```

$a[2]=0$

6: i := h;

$a[2]=0 \wedge h=1 \wedge i=h$

11: i := i+1;

$a[2]=0 \wedge h=1 \wedge i=2$

13: v := a[i];

$h=1 \wedge i=2 \wedge v=0 \wedge h \leq j \wedge j \geq 1$

20: j := j - h;

$h=1 \wedge i=2 \wedge v=0 \wedge j=0$

23: a[j] := v;

$a[0]=0$

$x > 0$

# Some Related Approaches

- Bug-Assist [Jose, Majumdar, ‘11]
- Whodunit? [Wang, Yang, Ivancic, Gupta, ’06]
- Delta debugging [Cleve, Zeller, ‘05]
- Distance metrics [Groce, Kroening, Lerda, ’04]

# Summary (Part 1)

Error invariants:

- new approach to fault localization
- enables computation of concise error explanations
- underlying work horse: Craig interpolation

**Part 2**

**Computing Craig Interpolants:**

Hierarchical Interpolation Procedures

joint work with Nishant Totla (IIT Bombay, India)

# Computing Craig Interpolants

Craig interpolants

- have many applications in Formal Methods
- can be automatically computed from proofs of unsatisfiability
- typically interested in ground interpolants
- many standard theories of SMT solvers admit ground interpolation
  - linear arithmetic
  - free function symbols with equality
  - ... (more tomorrow)

# Challenges in Interpolation

- Formulas generated in program verification are often in theories that are not directly supported by SMT solvers.
- Instead these theories are encoded using axioms and instantiation heuristics.
- Sometimes these heuristics are complete: (hierarchical decision procedure, SMT modulo theories).

How can we compute ground interpolants for such theory extensions (hierarchical interpolation)?

# Challenges in Interpolation

- We are not interested in arbitrary interpolants but only in inductive ones.
- Different proofs of unsatisfiability produce different interpolants.
- Finding *good* interpolants requires a lot of heuristics in the interpolation procedure.
- This is considered a black art.

How can we decouple these heuristics from the actual interpolation procedure?

# Example: List Reversal

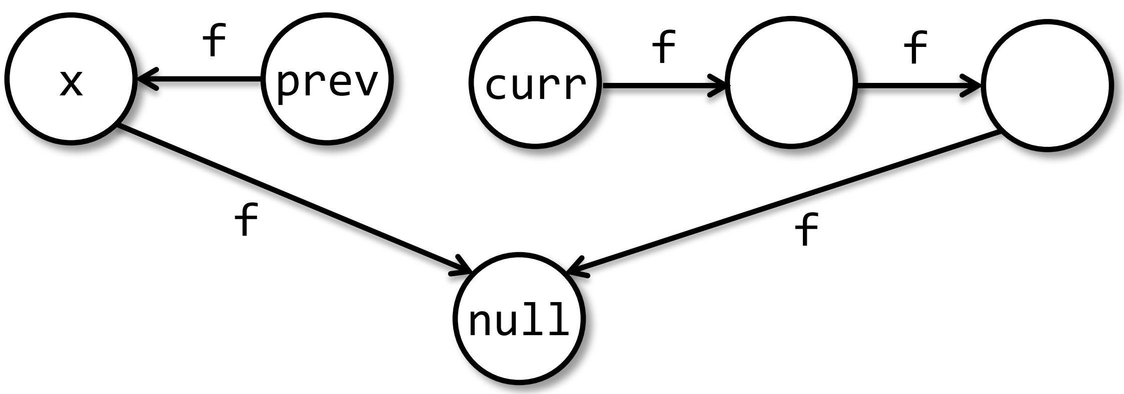
```
assume x -f-> null;
prev := null;
curr := x;
while curr ≠ null do
  succ := curr.f;
  curr.f := prev;
  prev := curr;
  curr := succ;
end
x := prev;
assert x -f-> null;
```

Safe inductive invariant:

$\mathtt{prev} \xrightarrow{\mathtt{f}} \mathtt{null} \wedge$

$\mathtt{curr} \xrightarrow{\mathtt{f}} \mathtt{null} \wedge$

$\mathrm{disjoint}(\mathtt{f}, \mathtt{curr}, \mathtt{prev})$



# Theory of Linked Lists with Reachability

(Variation of [Lahiri, Qadeer, POPL'08])

- $x \xrightarrow{f/u} y$ constrained reachability
- x.f field access: select(f,x)
- f [x := y] field update: update(f, x, y)

$x \xrightarrow{f/u} y$ means y is reachable from x via f without going through u (but y = u is allowed)

$x \xrightarrow{f} y$ stands for $x \xrightarrow{f/y} y$

# Axioms of TLLR

- Refl: $x \xrightarrow{f/u} x$
- Step: $x \xrightarrow{f/u} x.f \lor x{=}u$
- Linear: $x \xrightarrow{f} y \Rightarrow x \xrightarrow{f/y} u \lor x \xrightarrow{f/u} y$
- ...
- ReadWrite1: $x.(f[x := y]) = y$
- ReadWrite2: $x = y \lor y.(f[x := z]) = y.f$
- ReadWrite: $x \xrightarrow{f[u := v]/w} y \Leftrightarrow$

  $x \xrightarrow{f/w} y \land x \xrightarrow{f/u} y \lor$

  $x \xrightarrow{f/w} u \land v \xrightarrow{f/u} y \land v \xrightarrow{f/w} y \land u \neq w$

# Example Proof

$a \xrightarrow{f} b \wedge b.f = c \wedge \neg\, a \xrightarrow{f} c$

1. $a \xrightarrow{f} b$
2. $b.f = c$
3. $\neg\, a \xrightarrow{f} c$
4. $b \xrightarrow{f} b.f$ (Instantiation of Step)
5. $a \xrightarrow{f} b \wedge b \xrightarrow{f} c \Rightarrow a \xrightarrow{f} c$ (Inst. of Trans)
6. $b \xrightarrow{f} c$ (Rewriting of 4 with 2)
7. $\perp$ (Resolution of 5 with 1,2,6)

# Local Theory Extensions

[Sofronie-Stokkermans '05]

Given

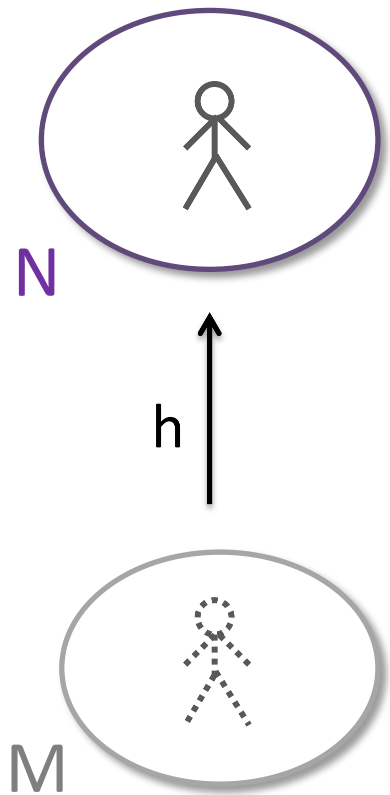
- a first-order signature $\Sigma_0$
- a (universal) first-order theory $T_0$ over $\Sigma_0$ (base theory)
- an extended signature $\Sigma_1 = \Sigma_0 \cup \{f_1,\ldots,f_n\}$
- a (universal) theory extension $T_1 = T_0 \cup K$

$T_1$ is called local if for all ground $\Sigma_1$-formulas G

$$T_1 \wedge G \models \bot \text{ iff } K[G] \wedge T_0 \wedge G \models \bot$$

Local extensions of decidable base theories are decidable.

# Detecting Locality
[Sofronie-Stokkermans ’05]



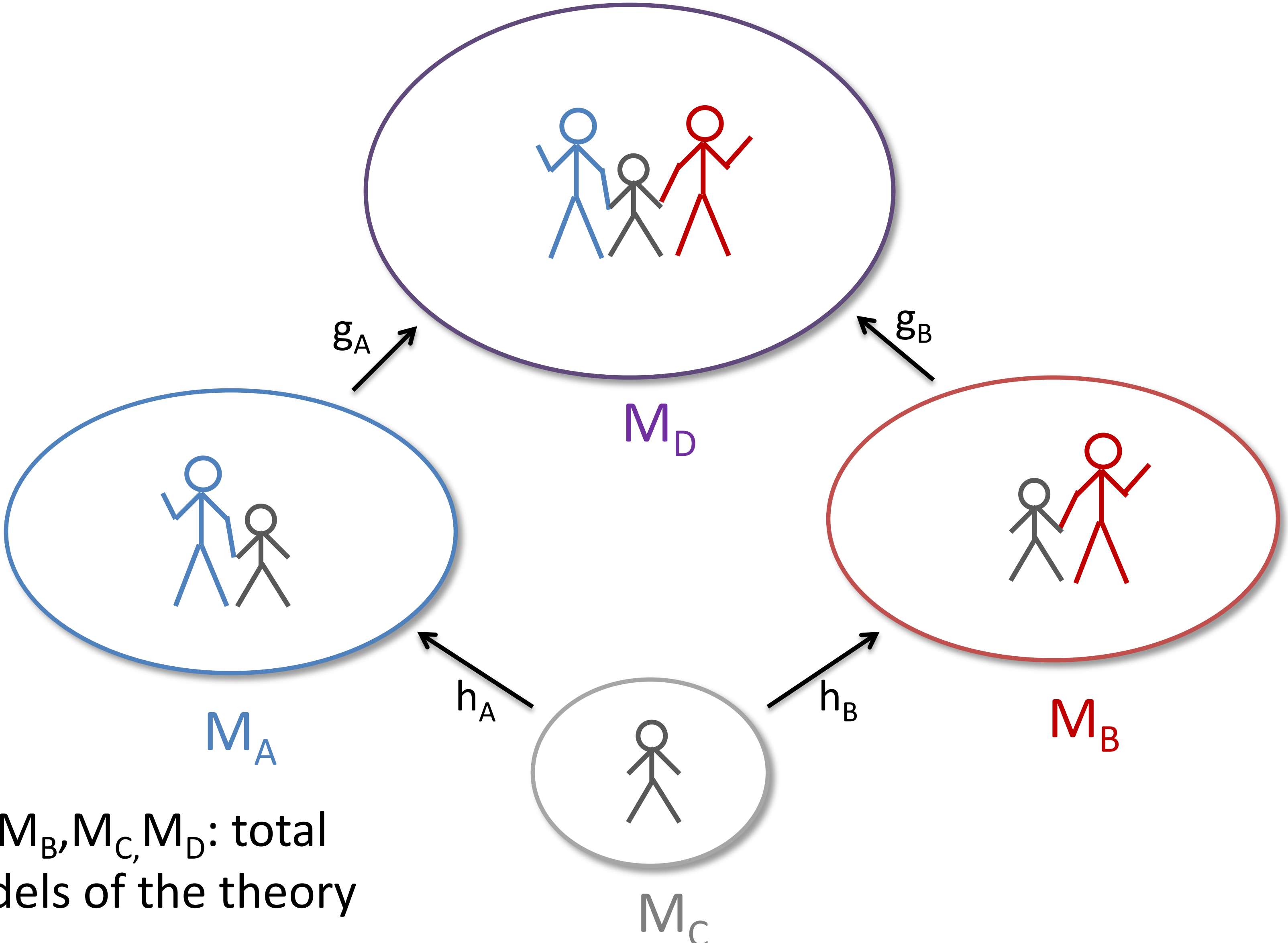

M: weak partial model of the theory
N: total model of the theory
h: homomorphic embedding of M into N

# Idea of Hierarchical Interpolation

Reduce interpolation problem $A \wedge B$ in theory extension $T_1 = K \cup T_0$ to interpolation problem in the base theory $T_0$:

- In order to find an $T_1$-interpolant $I$ for $A \wedge B$
- find a $T_0$-interpolant $I$ for $A_0 \wedge B_0$ where $A_0 = K[A] \wedge A$ and $B_0 = K[B] \wedge B$
- This is complete whenever for all $A \wedge B$ $T_1 \wedge A \wedge B \models \perp$ iff $T_0 \wedge A_0 \wedge B_0 \models \perp$

# Amalgamation Property




$M_A$,$M_B$,$M_C$,$M_D$: total models of the theory

# Amalgamation Property

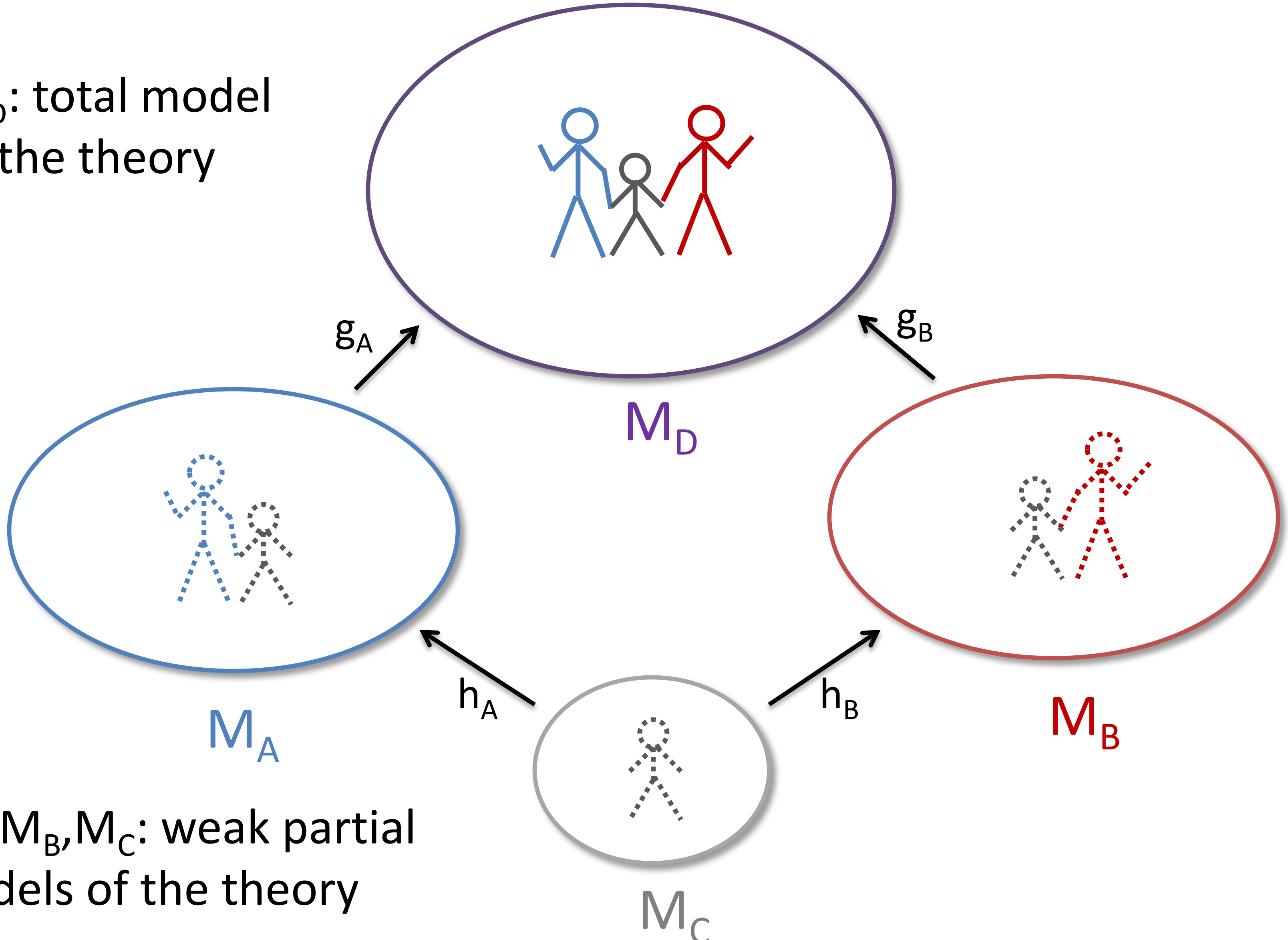
- A theory T admits ground interpolation iff T has the amalgamation property [Bacsich ‘75]

- Amalgamation does not tell us how to compute ground interpolants.

- Also, this property is too weak for our purposes.

# Weak Amalgamation Property

$M_D$: total model of the theory




$M_A, M_B, M_C$: weak partial models of the theory

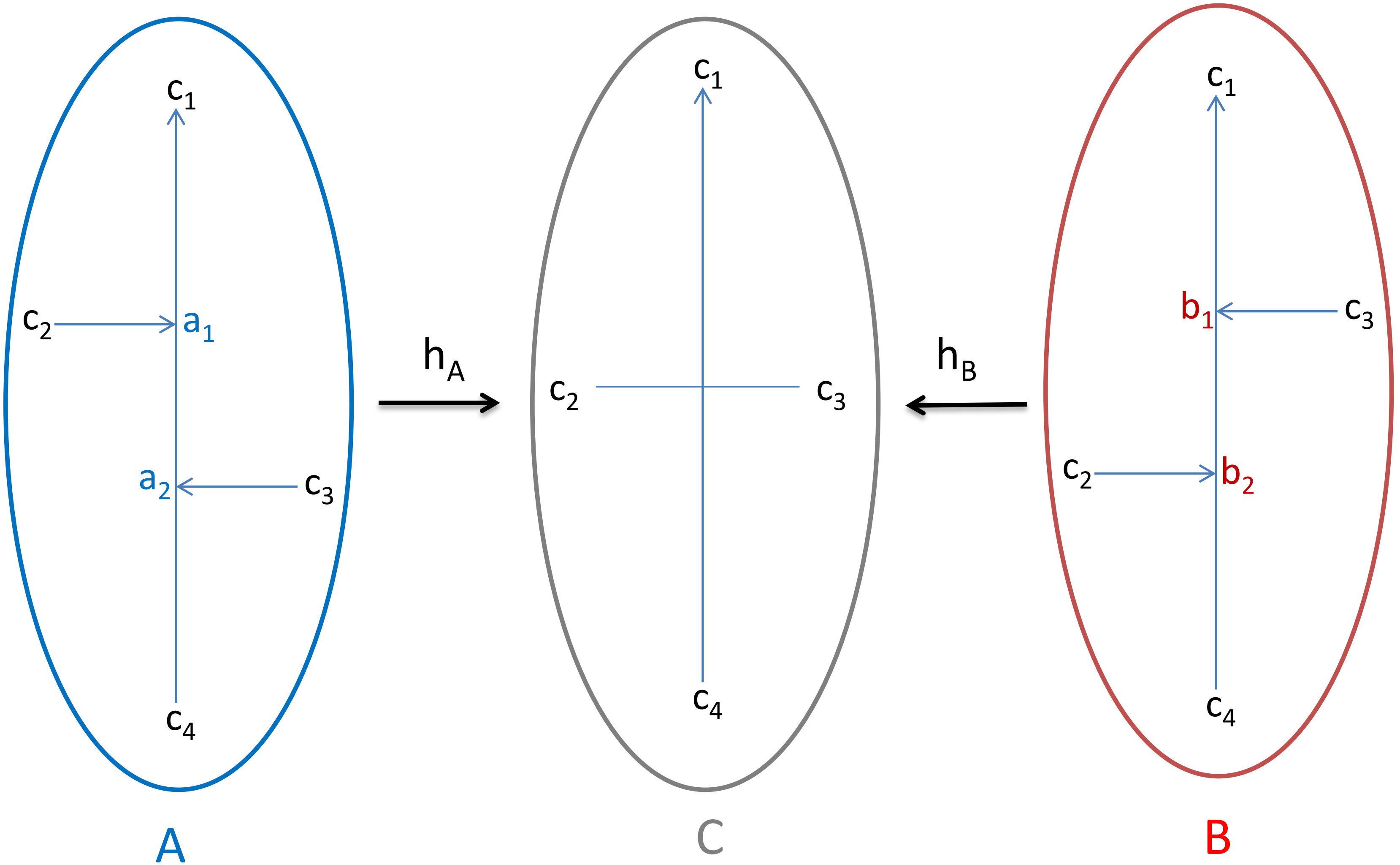
# Hierarchical Interpolation via Weak Amalgamation

**Main Result:**

If $T_1 = T_0 \cup K$ has the weak amalgamation property and $T_0$ admits effective ground interpolation, then $T_1$ admits effective ground interpolation.

Generic technique to obtain new interpolation procedures from existing ones.

# TLLR does not have weak amalgamation



# Making TLLR complete

Add two additional functions:

- join(f,x,y) — node where f-paths from x and y join (if they do so)
- diff(f,g) — node on which fields f and g differ if $f \neq g$

$x \xrightarrow{f} z \wedge y \xrightarrow{f} z \Rightarrow \text{join}(f,x,y) \xrightarrow{f} z$

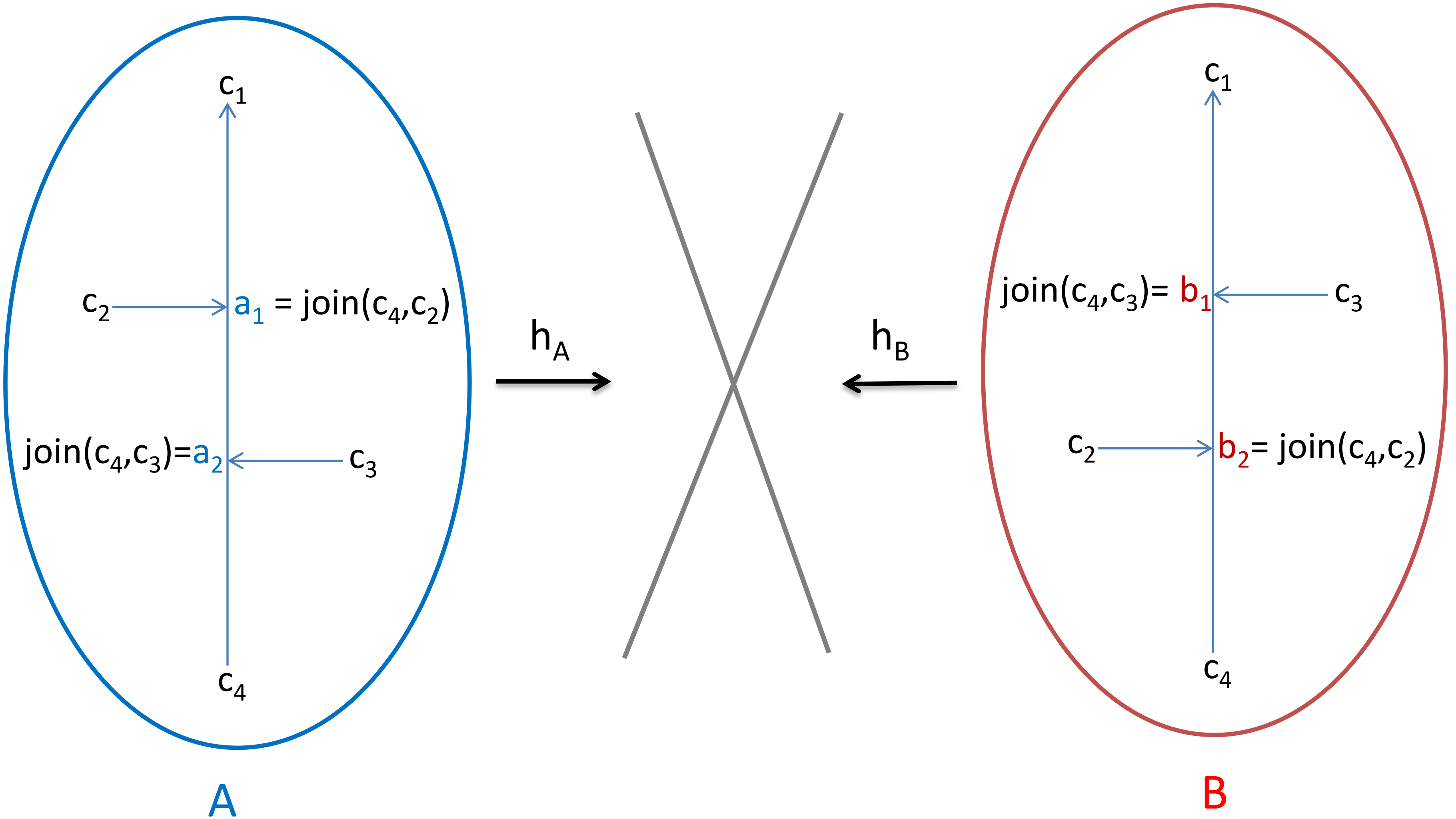
$x \xrightarrow{f} z \wedge y \xrightarrow{f} z \Rightarrow x \xrightarrow{f} \text{join}(f,x,y)$

$x \xrightarrow{f} z \wedge y \xrightarrow{f} z \Rightarrow y \xrightarrow{f} \text{join}(f,x,y)$

$\text{diff}(f,g).f = \text{diff}(f,g).g \Rightarrow f = g$

$$\text{disjoint}(f,x,y) \equiv x \xrightarrow{f} \text{join}(f,x,y) \wedge y \xrightarrow{f} \text{join}(f,x,y) \Rightarrow \text{join}(f,x,y) = \text{null}$$

# TLLR + join/diff has weak amalgamation



# Example: List Reversal

**assume** $x \xrightarrow{f} null$;
prev := null;
curr := x;
**while** curr $\neq$ null **do**
  succ := curr.f;
  curr.f := prev;
  prev := curr;
  curr := succ;
**end**
x := prev;
**assert** $x \xrightarrow{f} null$;

Safe inductive invariant:

$prev \xrightarrow{f} null \wedge$
$curr \xrightarrow{f} null \wedge$
disjoint(f,curr,prev)

Computed interpolant for
2 loop unrollings:

$(curr = null \wedge prev.f.f = null) \vee$
$(curr \neq null \wedge prev.f \neq curr \wedge$
$prev.f \neq curr.f \wedge curr.f \neq prev \wedge$
$prev.f.f = null)$

# Enumerating Partial Models

Given: theory extension T with weak amalgamation.

**Input**: A, B : ground formulas with $T \wedge A \wedge B \models \perp$

**Output**: I : $T_1$-interpolant for (A,B)

I := $\perp$

**while** $\exists$ partial model M of $T \wedge A \wedge \neg$ I **do**

    I := I $\vee$ interpolate(T, M, B)

**end**

**return** I

# Combining Interpolation and Abstraction

Given: theory extension T with weak amalgamation.

**Input**: A, B : ground formulas with $T \wedge A \wedge B \models \perp$

**Output**: I : $T_1$-interpolant for (A,B)

I := $\perp$

**while** $\exists$ partial model M of $T_0 \cup K[A] \wedge A \wedge \neg$ I **do**

    **if** $T \wedge \alpha(M) \wedge B \models \perp$ **then** // $M \models \alpha(M)$

        I := I $\vee$ interpolate(T, $\alpha(M)$, B)

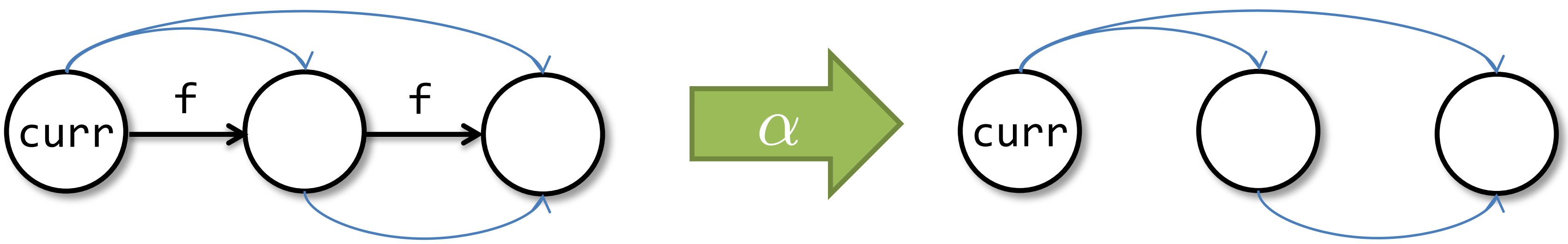
    **else**

        I := I $\vee$ interpolate(T, M, B)

**end**

**return** I

# List Abstraction




Abstract from the length of the list.

# Example: List Reversal

```
assume x ⟶f null;
prev := null;
curr := x;
while curr ≠ null do
  succ := curr.f;
  curr.f := prev;
  prev := curr;
  curr := succ;
end
x := prev;
assert x ⟶f null;
```

Safe inductive invariant:

$\mathtt{prev} \xrightarrow{f} \mathtt{null} \wedge$
$\mathtt{curr} \xrightarrow{f} \mathtt{null} \wedge$
disjoint(f,curr,prev)

Computed interpolant for 2 loop unrollings:

$\mathtt{prev} \xrightarrow{f/\mathtt{curr}} \mathtt{null} \wedge$
join(f,prev,curr) = null

# Related Work

- Sofronie-Stokkermans ‘06: Interpolation in local theory extensions
- Rybalchenko, Sofronie-Stokkermans ‘07: Constraint Solving for Interpolation
- Bruttomesso, Ghilardi, Ranise ‘12: Strong amalgamation for interpolation in theory combinations
- Bruttomesso, Ghilardi, Ranise ‘11: Ground interpolation for arrays
- McMillan ‘08: Quantified interpolation

# Summary (Part 2)

- A new generic technique to obtain new interpolation procedures from existing ones
  - depends only on a model theoretic notion (weak amalgamation)
  - interpolation procedure for base theory can be treated as a black box
  - allows easy implementation of domain-specific heuristics
- Many theories of practical interest have weak amalgamation
  - arrays with extensionality
  - linked lists with reachability
  - imperative trees with reachability [CADE’11]
  - ...