Source-tracking Unification*

Venkatesh Choppella
mailto:choppellav@ornl.gov
Oak Ridge National Laboratory, USA

Christopher T. Haynes
mailto:chaynes@cs.indiana.edu
Indiana University, USA

Aug 2nd, 2003

*CADE-19: Conference on Automated Deduction Jul 28th-Aug 2nd, 2003, Miami, USA. Research supported by the National Science Foundation, grant NSF-CDA-9312614 and the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC for the U. S. Department of Energy under contract number DE-AC05-00OR22725.

Source-tracking Unification
What is it and why is it useful?

Track the results of unification or its failure in terms of the source, i.e., the original presentation of problem

Automatic explanation and debugging

- Type Error slicing in Functional Programs
- Debugging Logic Programs

Introduction

Example: Type Error Slicing

Framework

Example Wrap up: Program Slicing

Related Research

Future Work and Conclusions

Different views of Solving Unification

Transforming Equations to solved form

Closure computation on Unification Graphs

Context Free Language Reachability

- Maximally Unifiable subsets [Cox, 1984]
- Systems of Logic [Le Chenadec, 1989]

Main Ideas

Unification Graphs are Labeled Directed Graphs

- Equational Edges are labeled epsilon
- Projection Edges labeled by projection symbols $\{f_i | f \in \Sigma, 1 \leq i \leq \mathit{arity}(f)\}$

Downward closure is like matching parentheses (semi-Dyck languages)

Membership in unification closure can be witnessed by semi-Dyck labeled paths (unification paths)

Explanations can be formulated as proofs in a type system

Proofs can be encoded as unification paths

Computation and simplification of these proofs is easy

Summary of Results

Model for characterizing unification source-tracking as reachability via semi-Dyck labeled paths

Type System for unification

Algorithm for computing shortest proofs in $O(V^3)$ time for fixed signature

Integration of proof generation with the unification algorithm

Simplification of proofs by elementary rewriting

Unfriendly Type Error messages

`Standard ML of New Jersey, Version 110.0.7, September 28, 2000`

$\lambda\ x.$
if x **then**
inc x // inc: *num* $\rightarrow$ *num*
else x;

```
Error:  case object and rules don't agree [tycon mismatch]
rule domain:bool,  object:num  in expression:
case x of
true => inc x
| false => x
```

Too much information!

Type Inference and Unification

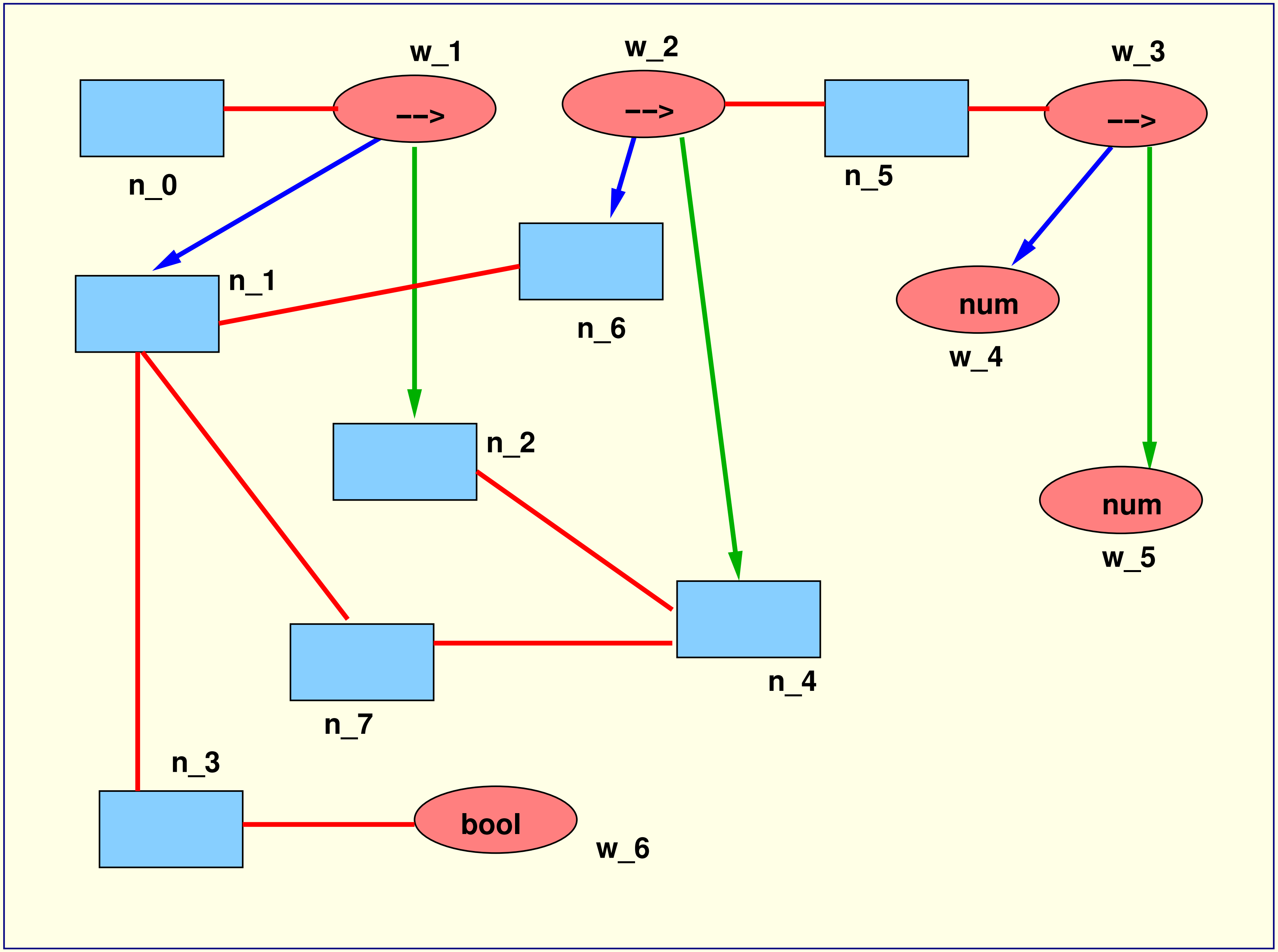
$$\boxed{\lambda_0\ x_1.\ \mathbf{if}_2\ x_3\ \mathbf{then}\ @_4(\mathit{inc}_5\ x_6)\ \mathbf{else}\ x_7}$$

Syntax Equations $\longrightarrow$ sets of Type Equations

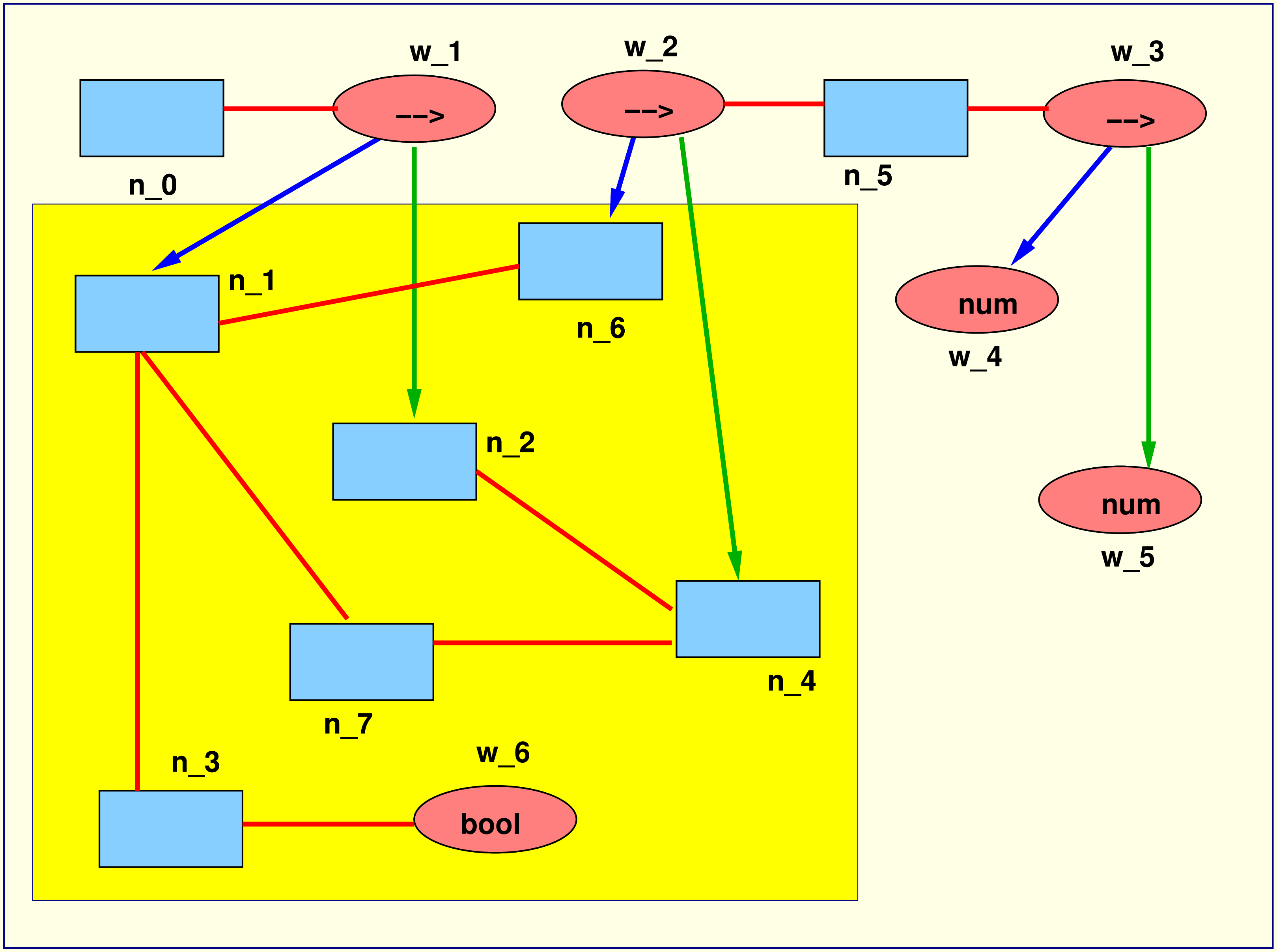
$$
\begin{array}{lcl}
n_0 = \lambda(n_1, n_2) & \mapsto & \{n_0 \stackrel{?}{=} n_1 \rightarrow n_2\} \\
n_2 = \mathsf{if}(n_3, n_4, n_7) & \mapsto & \{n_3 \stackrel{?}{=} \mathsf{bool}, n_2 \stackrel{?}{=} n_4, \\
 & & n_4 \stackrel{?}{=} n_7\} \\
n_3 = \lambda\mathsf{var}(n_1) & \mapsto & \{n_3 \stackrel{?}{=} n_1\} \\
n_4 = @(n_5, n_6) & \mapsto & \{n_5 \stackrel{?}{=} n_6 \rightarrow n_4\} \\
n_5 = \mathsf{const}(\mathit{num} \rightarrow \mathit{num}) & \mapsto & \{n_5 \stackrel{?}{=} \mathsf{num} \rightarrow \mathsf{num}\} \\
n_6 = \lambda\mathsf{var}(n_1) & \mapsto & \{n_6 \stackrel{?}{=} n_1\} \\
n_7 = \lambda\mathsf{var}(n_1) & \mapsto & \{n_7 \stackrel{?}{=} n_1\}
\end{array}
$$

Computing Unification Closure

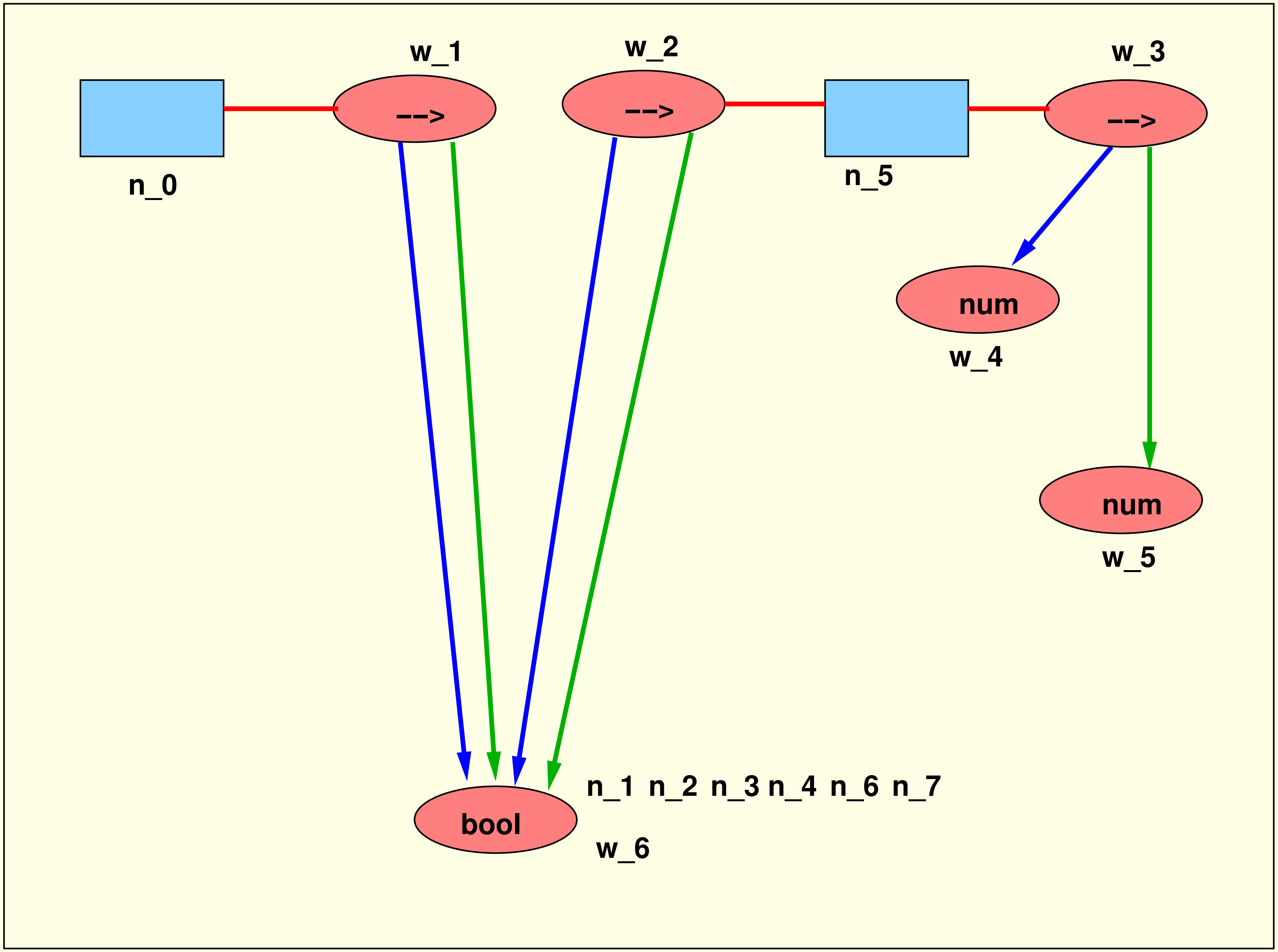
Computing Unification Closure



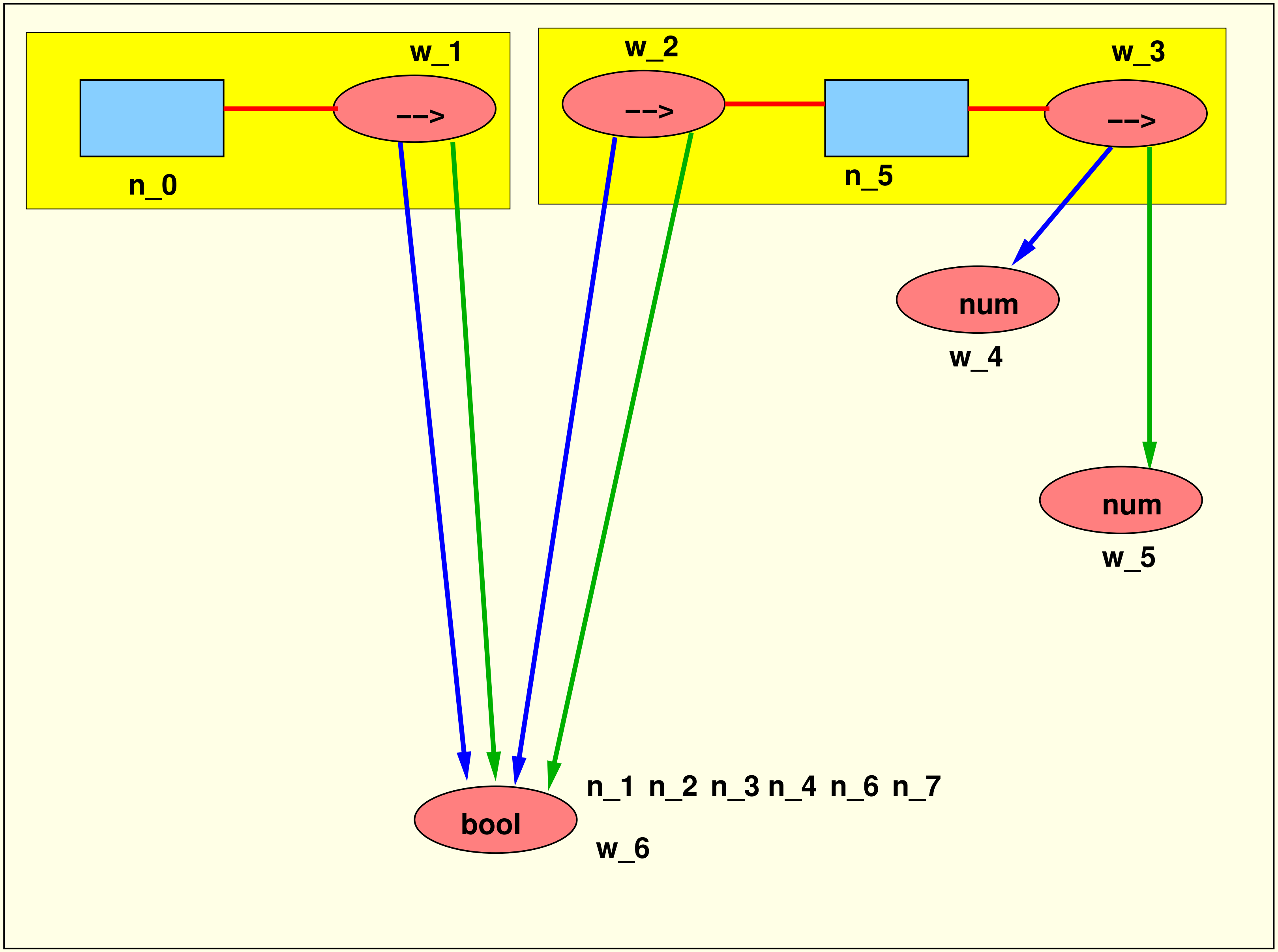
Computing Unification Closure



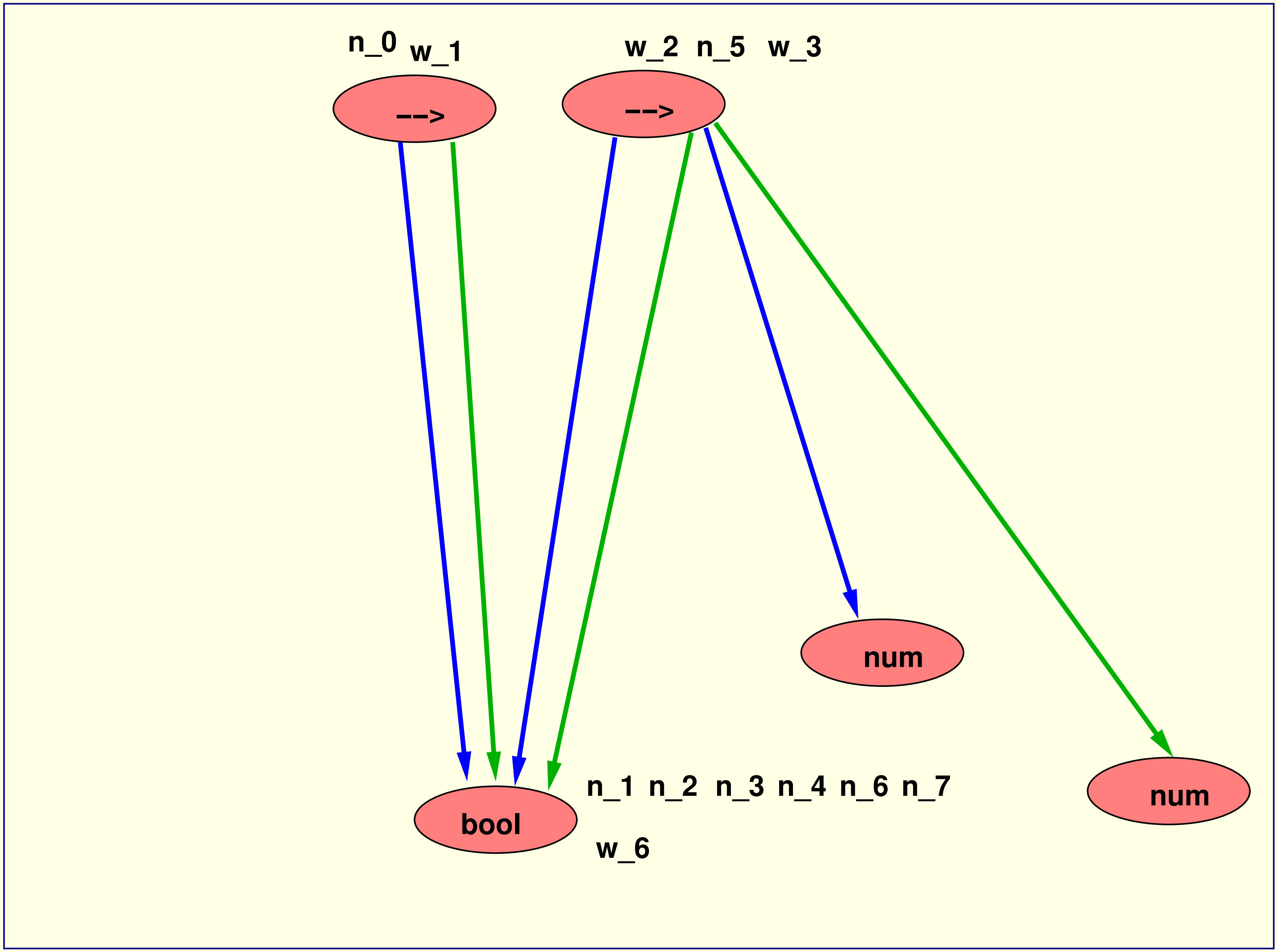
Computing Unification Closure



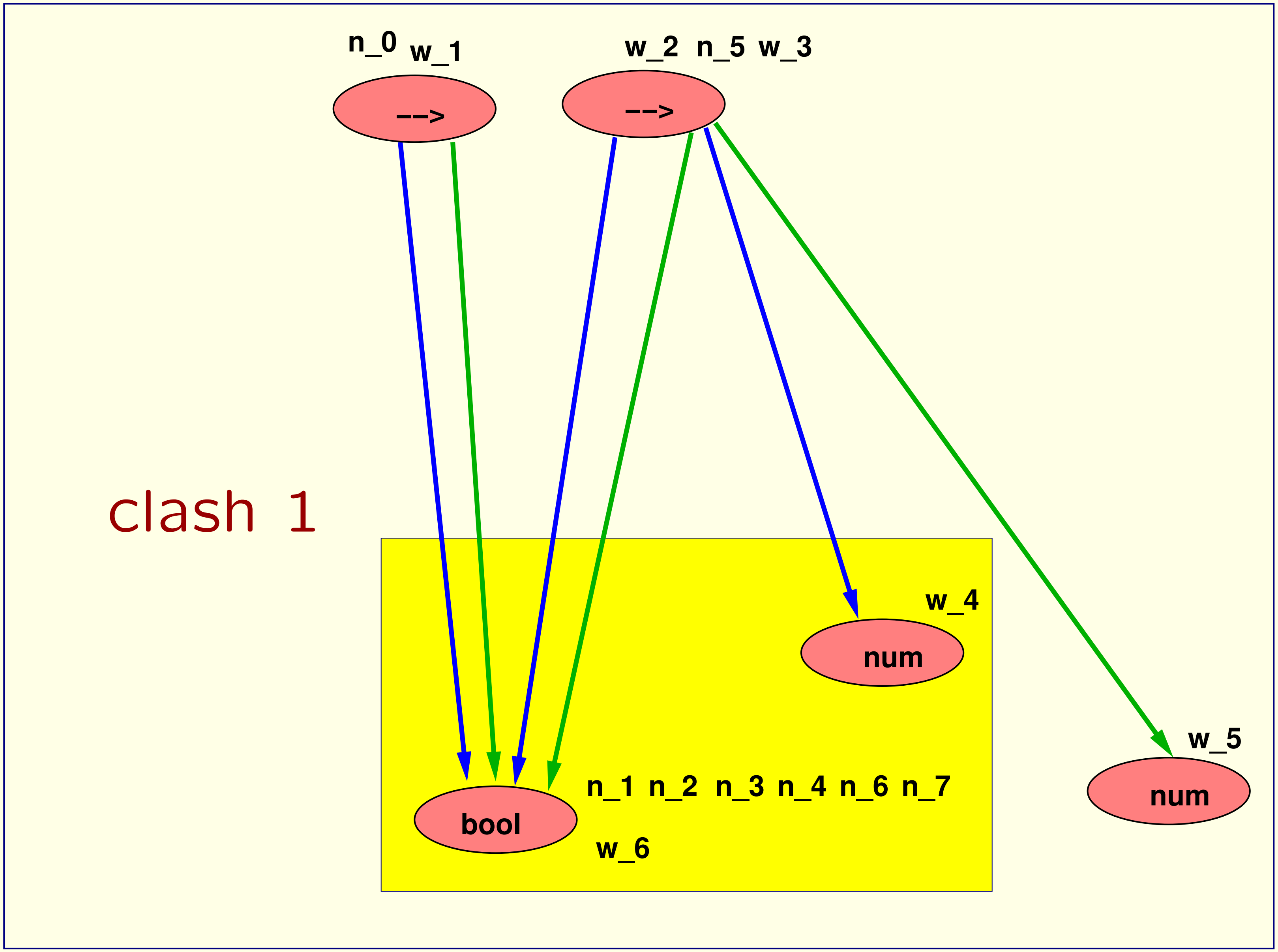
Computing Unification Closure



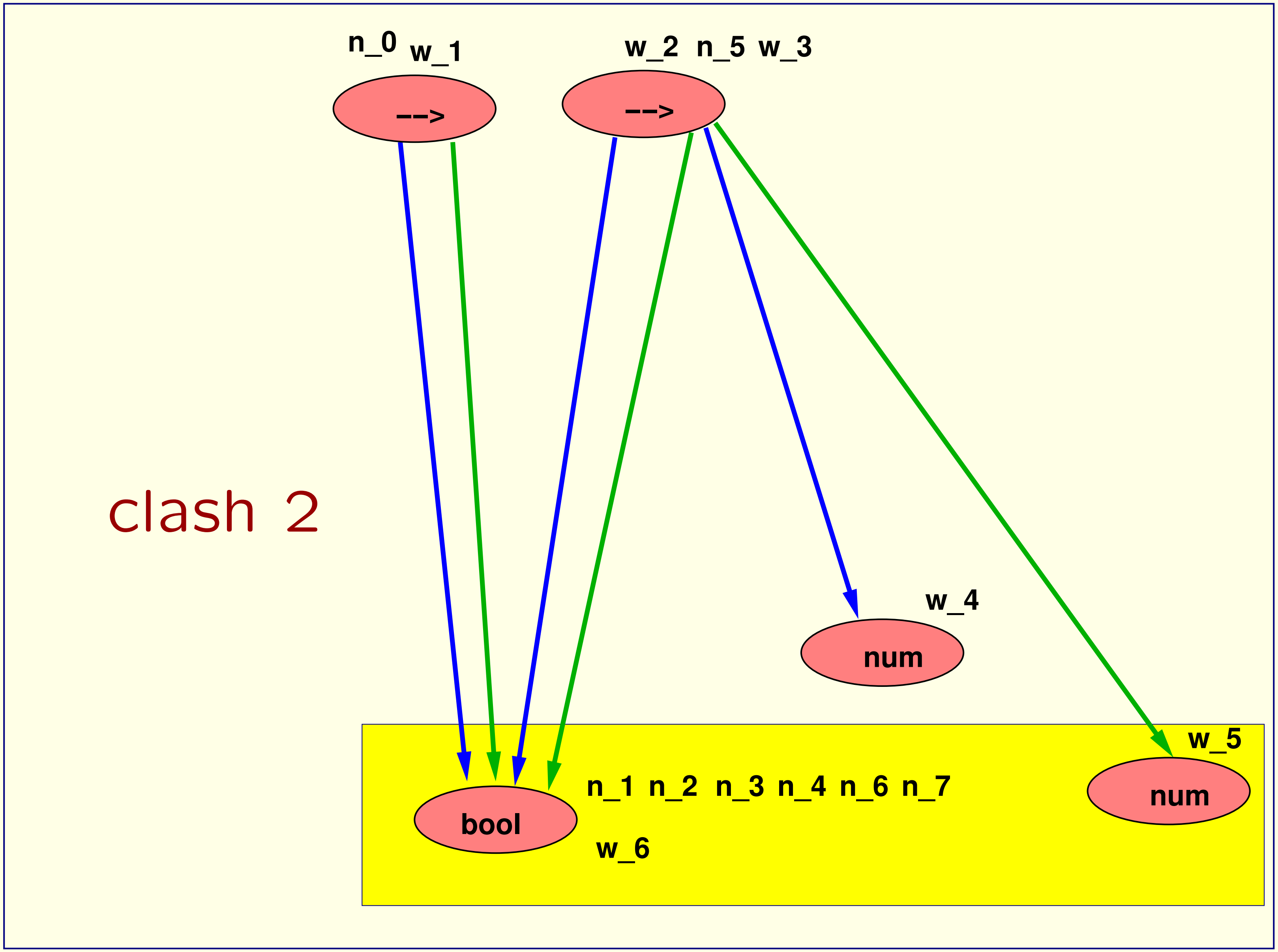
Computing Unification Closure



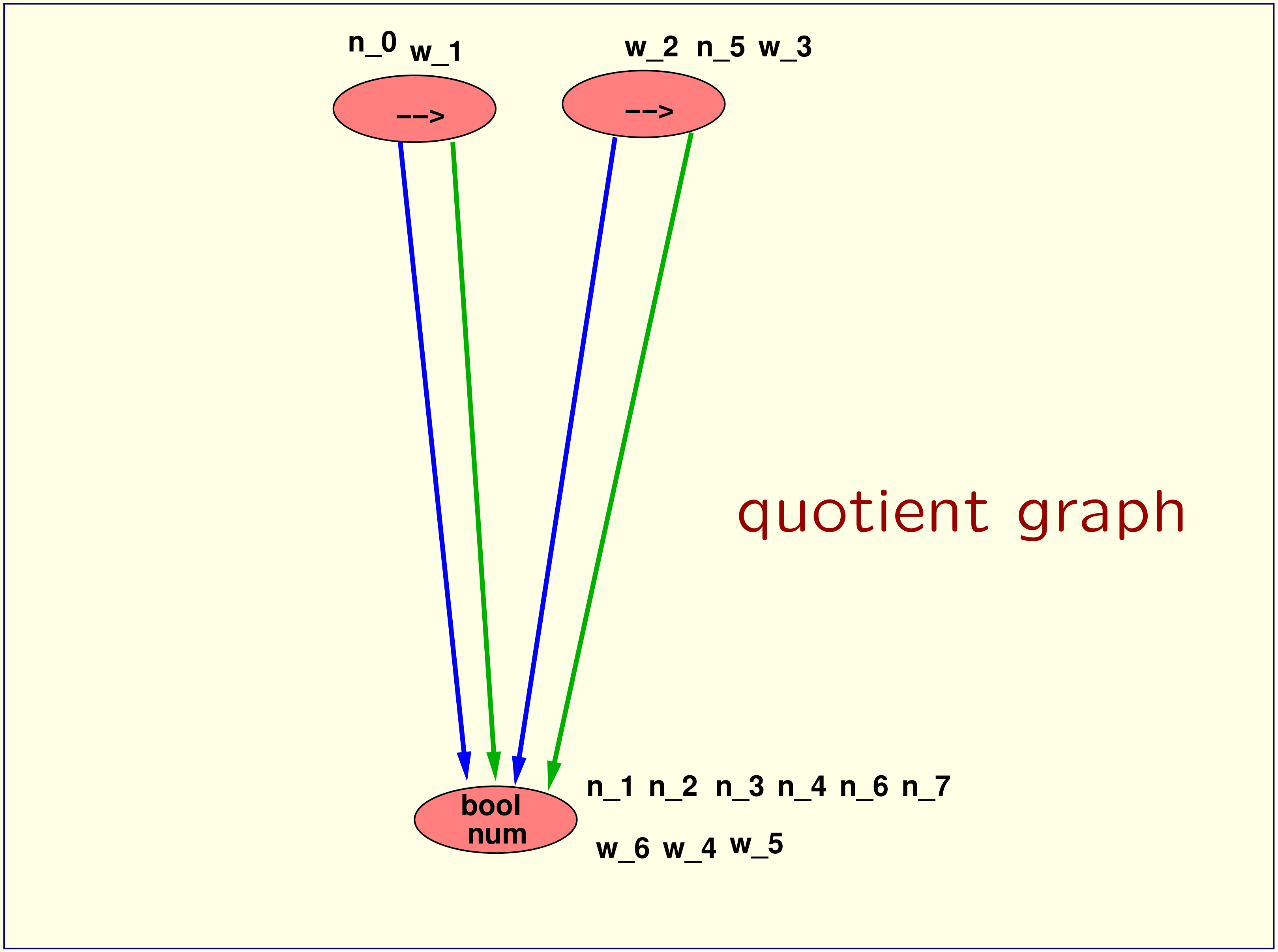
Computing Unification Closure



Computing Unification Closure



Computing Unification Closure



The Problem of Source-tracking Unification

G unifiable iff the quotient of G is clash and cycle free.

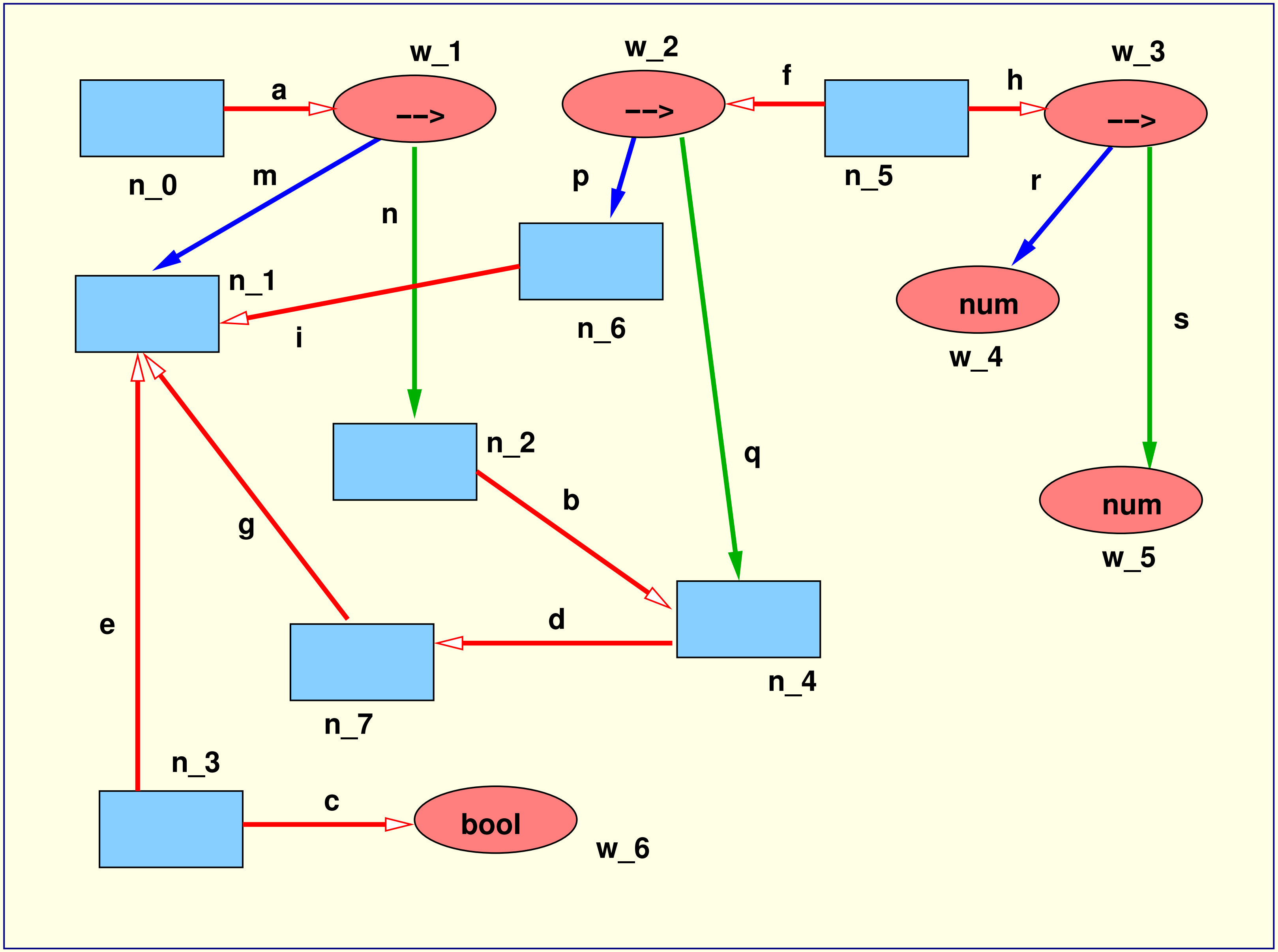
[Paterson and Wegman, 1978]

How to express connectivity of the quotient of G ...

in terms of the connectivity of G

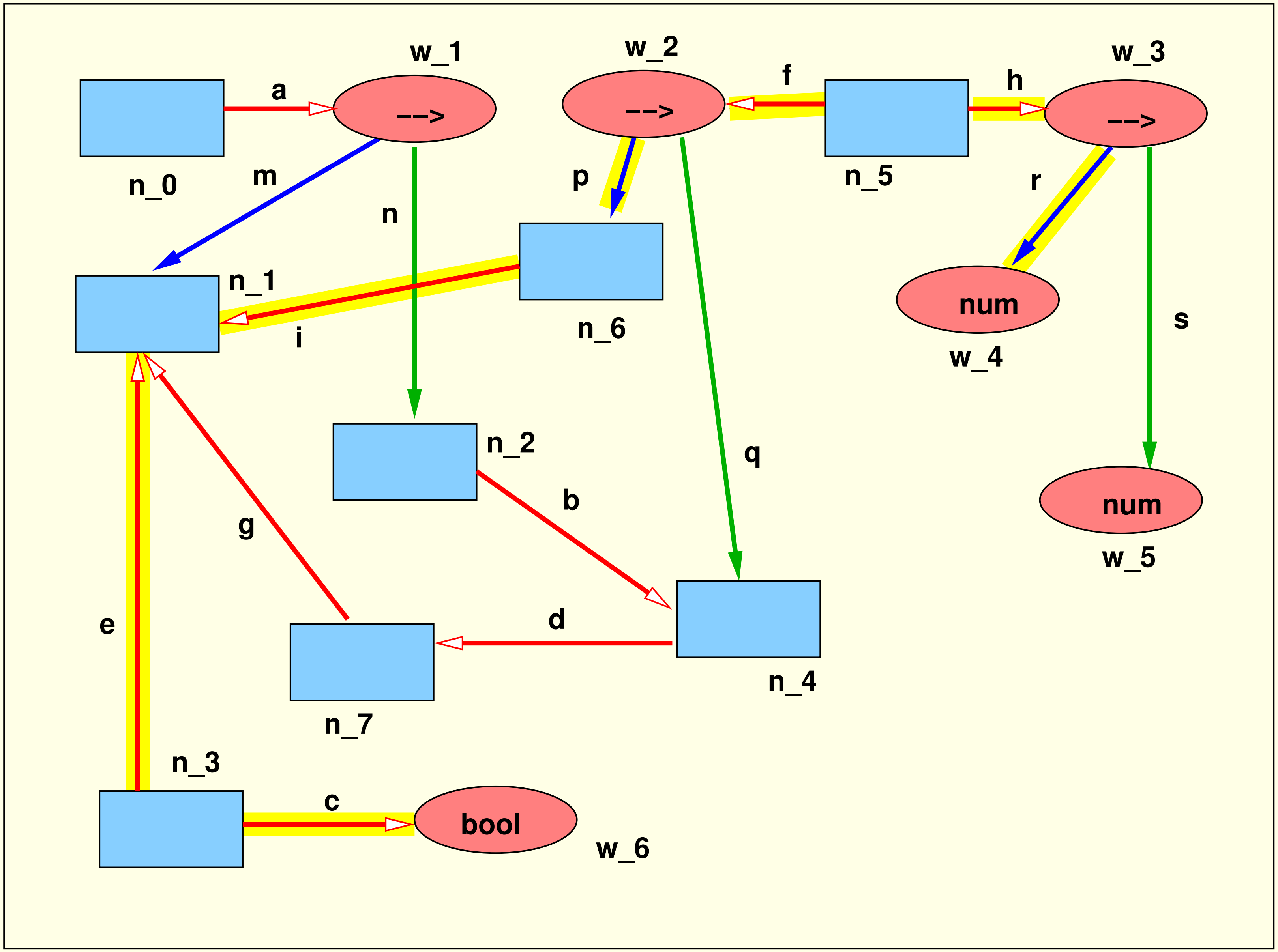
Clash 1

Clash 1

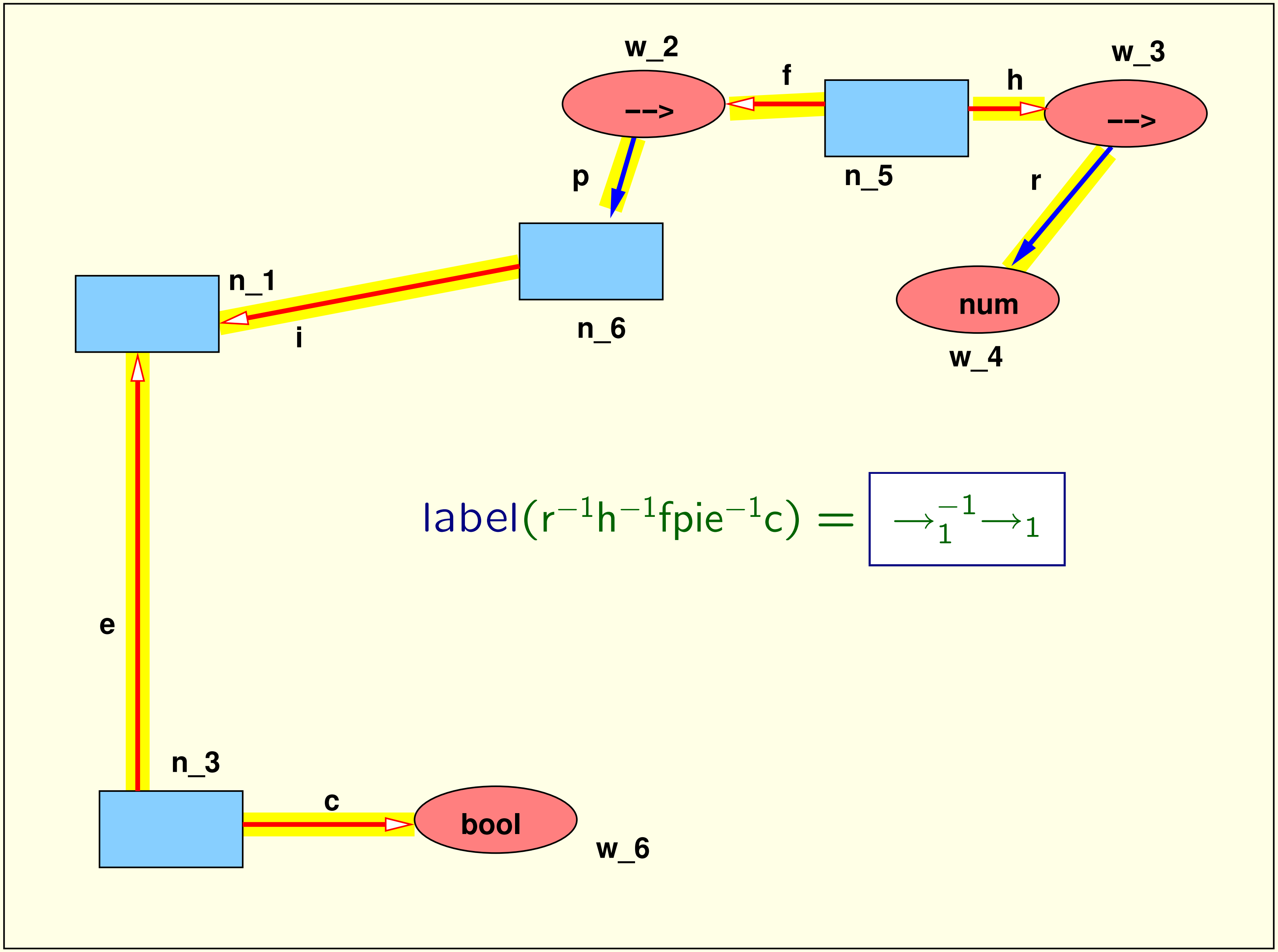


w_1
w_2
w_3
a
f
h
-->
-->
-->
n_0
m
p
n_5
r
n
n_1
n_6
num
s
i
w_4
n_2
q
b
num
g
w_5
e
d
n_4
n_7
n_3
c
bool
w_6

Clash 1


w_1
w_2
w_3
a
f
h
-->
-->
-->
n_0
m
p
n_5
r
n
n_1
n_6
num
w_4
s
i
n_2
q
b
num
w_5
g
e
d
n_4
n_7
n_3
c
bool
w_6

Clash 1


w_2
f
h
w_3
-->
-->
p
n_5
r
n_1
n_6
num
i
w_4
e
n_3
c
bool
w_6
$\mathrm{label}(r^{-1}h^{-1}fpie^{-1}c) = \boxed{\longrightarrow_1^{-1}\longrightarrow_1}$

Introduction

Example: Type Error Slicing

Framework

Example Wrap up: Program Slicing

Related Research

Future Work and Conclusions

Semi-Dyck Sets and Unification Paths

Σ: right parenthesis symbols

Σ^{-1}: left parenthesis symbols

Words over $\Sigma \cup \Sigma^{-1}$

One-way cancellation

$$\{\delta^{-1}\delta \longrightarrow \epsilon \mid \delta \in \Sigma\}$$

Unification Paths over G:

Paths in $G \cup G^{-1}$ whose labels normalize to $\boxed{\text{words over } \Sigma}$

Unification Source-tracking Theorem

Soundness

Unification path over G with label l implies path in the quotient of G with label equal to the normal form of l

Completeness

Path in the quotient of G with label l implies unification path over G with label whose normal form is l

Connectivity in the quotient of G expressed ...

... in terms of connectivity in $G \cup G^{-1}$

Computing Shortest Unification Paths

Computation of shortest unification path using CFL shortest path algorithm [Barrett et al. '00]

Can be computed in $O(V^3)$ for fixed alphabet.

$P^U(G)$: A Simple Type System for Unification

$G = \langle \Sigma, V, D \rangle$ labeled directed graph

$p \in T(\Sigma_{\mathbf{Gr}}, D)$: a free group term generated by the edges D

$u, v \in V$: vertices of G, $l \in \Sigma^*$: Word over Σ

Type judgements $\boxed{G \vdash p : u \xrightarrow{l} v}$

INIT	$$\frac{}{G \vdash c : u \xrightarrow{\delta} v}$$	$c : u \xrightarrow{\delta} v \in G$
REF	$$\frac{}{G \vdash \epsilon : u \xrightarrow{\epsilon} u}$$	$u \in G$
SYM	$$\frac{G \vdash p : v \xrightarrow{\epsilon} u}{G \vdash p^{-1} : u \xrightarrow{\epsilon} v}$$	
TRANS	$$\frac{G \vdash p : u \xrightarrow{l} v' \qquad G \vdash q : v' \xrightarrow{l'} v}{G \vdash pq : u \xrightarrow{ll'} v}$$	
DN	$$\frac{G \vdash p : w \xrightarrow{\epsilon} w'}{G \vdash c^{-1}pc' : u \xrightarrow{\epsilon} v}$$	$c : w \xrightarrow{\delta} u \in G$ $c' : w' \xrightarrow{\delta} v \in G$

Figure 1: The logic $P^U(G)$ of unification path expressions over G

$P^U(G)$ adequacy theorem

Let $G = \langle \boldsymbol{\Sigma}, V, D \rangle$ be a labeled directed graph.

1. (Soundness) If $G \vdash_{P^U} p : u \xrightarrow{l} v$, then $G/{\sim} \models u \xrightarrow{l} v$.

2. (Completeness) If $G/{\sim} \models u \xrightarrow{l} v$, then $G \vdash_{P^U} p : u \xrightarrow{l} v$ where p is some $\boldsymbol{\Sigma}_{\mathbf{Gr}}$-term over D.

Constructing Unification Proofs

procedure *unify*$(v_1, v_2, \boxed{m}) =$
  let $\langle r_1, \boxed{p_1}\rangle =$ *find*(v_1) **and** $\langle r_2, \boxed{p_2}\rangle =$ *find*(v_2)
    in if $r_1 = r_2$ **then return**
      else case r_1.*type*, r_2.*type*
        strict, strict: *union*$(r_1, r_2, \boxed{(p_1)^{-1} m p_2})$
        functor, strict: *unify*$(v_2, v_1, \boxed{(m)^{-1}})$
        strict, functor: **let** *ans* $=$ *occurs?*(r_2, r_1)
          in case *ans*
            no: *union*$(r_1, r_2, \boxed{(p_1)^{-1} m p_2})$
            yes$(_ \, , \; q)$: *fail*$(\boxed{\text{CYCLE}, (p_1)^{-1} m p_2 q})$

Unification algorithm with source-tracking: procedure *unify*

Simplification of Unification Proofs

Free group rewriting rules [Peterson and Stickel, '81]

Weak Subject Reduction

One-Step rewriting breaks types!

... but types reappear at normalization

Unification Source-tracking: Summary

Definition using unification paths

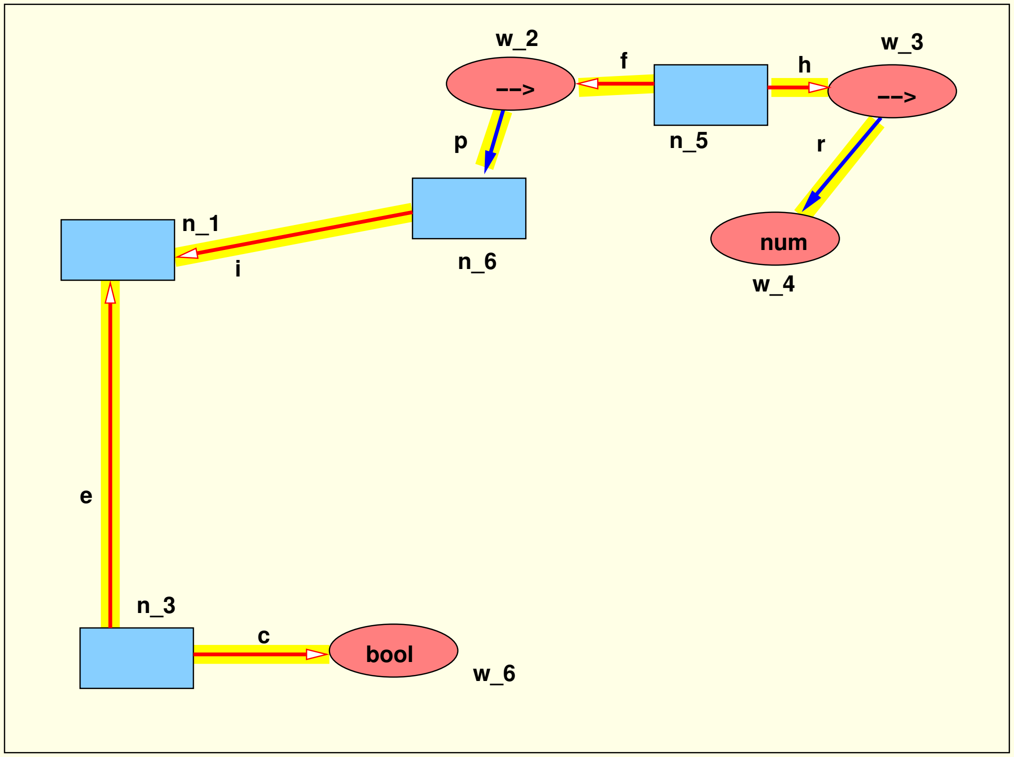
Optimization using shortest-path algorithms

Deduction using the Logic for unification path expressions

Construction using standard unification algorithms

Simplification using group rewriting

Type Equation Slice for clash 1



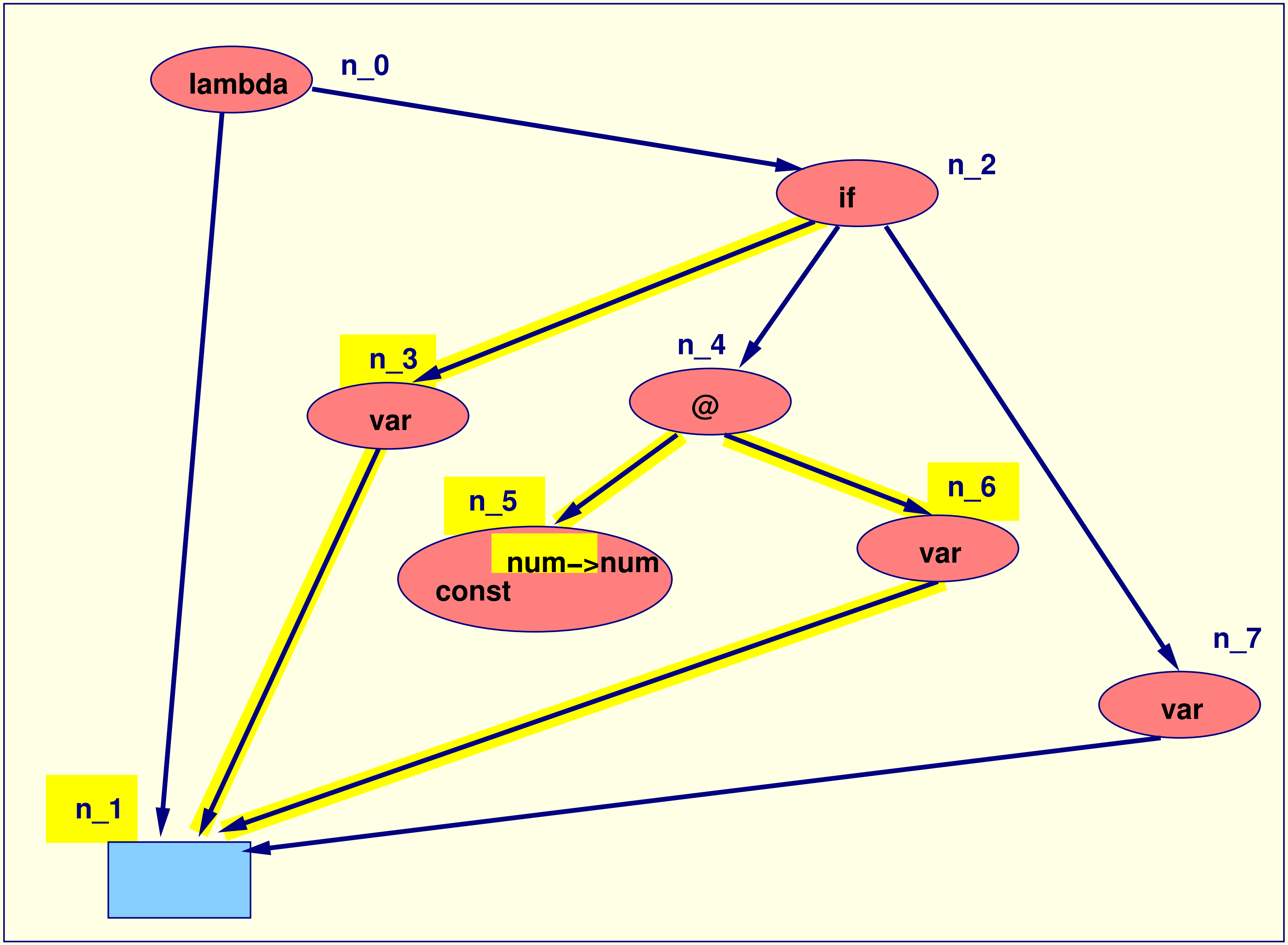
$$\begin{array}{lcl} \mathsf{bool} & \stackrel{?}{=} & n_3 \\ n_3 & \stackrel{?}{=} & n_1 \\ n_1 & \stackrel{?}{=} & n_6 \\ n_5 & \stackrel{?}{=} & n_6 \rightarrow \square \\ n_5 & \stackrel{?}{=} & \mathit{num} \rightarrow \square \end{array}$$

Program Slice for clash 1

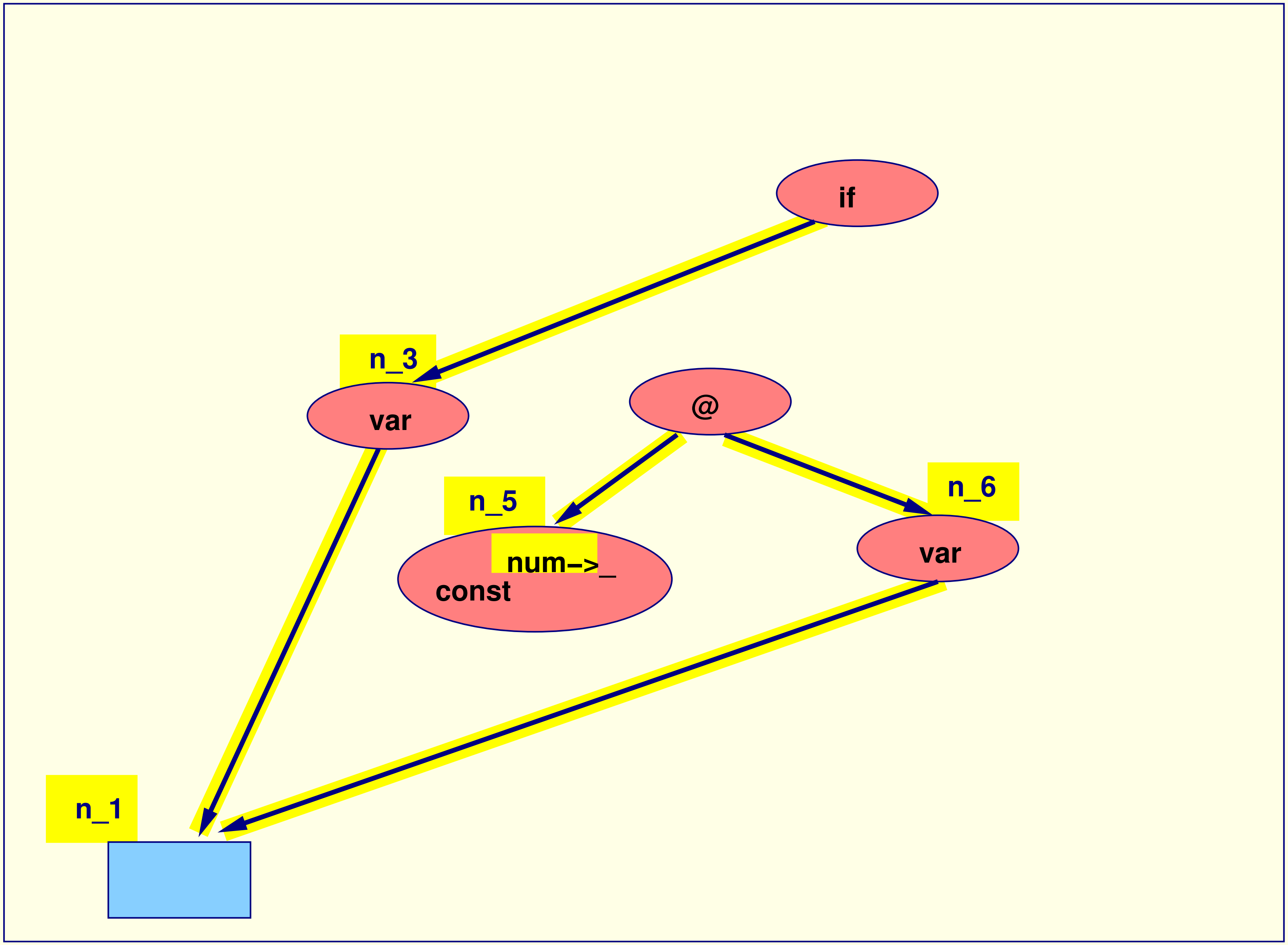
$$
\begin{array}{lcll}
\square & = & \mathsf{if}(n_3, \square, \square) & \mapsto \ \mathsf{bool} \overset{?}{=} n_3 \\
n_3 & = & \lambda\mathsf{var}(n_1) & \mapsto \ n_3 \overset{?}{=} n_1 \\
n_6 & = & \lambda\mathsf{var}(n_1) & \mapsto \ n_1 \overset{?}{=} n_6 \\
\square & = & @(n_5, n_6) & \mapsto \ n_5 \overset{?}{=} n_6 \rightarrow \square \\
n_5 & = & \mathsf{const}(\mathit{num} \rightarrow \square) & \mapsto \ n_5 \overset{?}{=} \mathit{num} \rightarrow \square
\end{array}
$$

Graph of Program Slice S_1

Graph of Program Slice S_1



Graph of Program Slice S_1



Related Research

Reason lists [Wand '86]

Not enough reasons accumulated to simulate error
Can't eliminating irrelevant reasons: lacks cancellative rules

Flow techniques [Johnson-Walz'86]

Error-tolerant unification
Complicated algorithm, informally stated

Explanation-based systems [Stansifer '94, Duggan '94]

Interactive graph navigation
Lack automation

Logic Programming

Maximally unifiable subsets [Cox,'84, Chen et al.'86]
Unification failure [Cox, '87, Port, '88]

Origin Tracking in Rewrite Systems [Bertot, '95, van Deursen et al. '93]

Future Work

Measurements

- Evaluate efficiency and output sizes of algorithms for realistic unification problems.
- How bad is the non-optimal algorithm in practice.
- How effective is simplification
- How to generate minimal proofs

Applications

- Diagnosis of errors in Hindley-Milner type inference
- Prolog debugging and backtracking

Extensions

- Semi-Unification (useful for Polymorphic Type Inference with Recursion)
- does this framework extend easily to other unification theories?

Conclusions

Relate unification with Path Problems

Simple Logic to compute well-formed "explanations"

Algorithms for computing and simplifying source-tracking information

Interactive generation of unification source-tracking information prototype implemented in Chez Scheme

$\texttt{http}:\texttt{//www.cs.indiana.edu/hyplan/chaynes/unif.tar.gz}$

Introduction

Example: Type Error Slicing

Framework

Example Wrap up: Program Slicing

Related Research

Future Work

Conclusions

Overflow

- Rewrite Rules for Path Simplification
- Subject Reduction Properties

Unification Closure $\sim_G$

EQ	$$\frac{}{u \sim v}$$	$u \xrightarrow{\epsilon} v \in G$
REF	$$\frac{}{u \sim u}$$	$u \in G$
SYM	$$\frac{v \sim u}{u \sim v}$$	
TRANS	$$\frac{u \sim v' \qquad v' \sim v}{u \sim v}$$	
DN	$$\frac{w \sim w'}{u \sim v}$$	$w \xrightarrow{f.i} u$ $w' \xrightarrow{f.i} v$

Path Simplification

Associativity	$(pq)r$	$=$	$p(qr)$	$=$	pqr
1	$(pq)^{-1}$	$\longrightarrow$	$q^{-1}p^{-1}$		
2	$(p^{-1})^{-1}$	$\longrightarrow$	p		
3	ϵ^{-1}	$\longrightarrow$	ϵ		
4	$p\epsilon$	$\longrightarrow$	p		
5	ϵp	$\longrightarrow$	p		
6	pp^{-1}	$\longrightarrow$	ϵ		
7	$p^{-1}p$	$\longrightarrow$	ϵ		

Figure 3: Equational rewrite system R/A for free groups, where A consists of the equational rule of aassociativity, and R consists of the remaining rules (Peterson and Stickel, 1981).

R/A is strongly normalizing. Reduction under R/A yields unique normal forms.

Subject Reduction Properties

Rewriting with R/A destroys P^U proofs ... temporarily.

Let

$$a : w \xrightarrow{\epsilon} w'$$

$$b_1 : w \xrightarrow{f.i} u$$

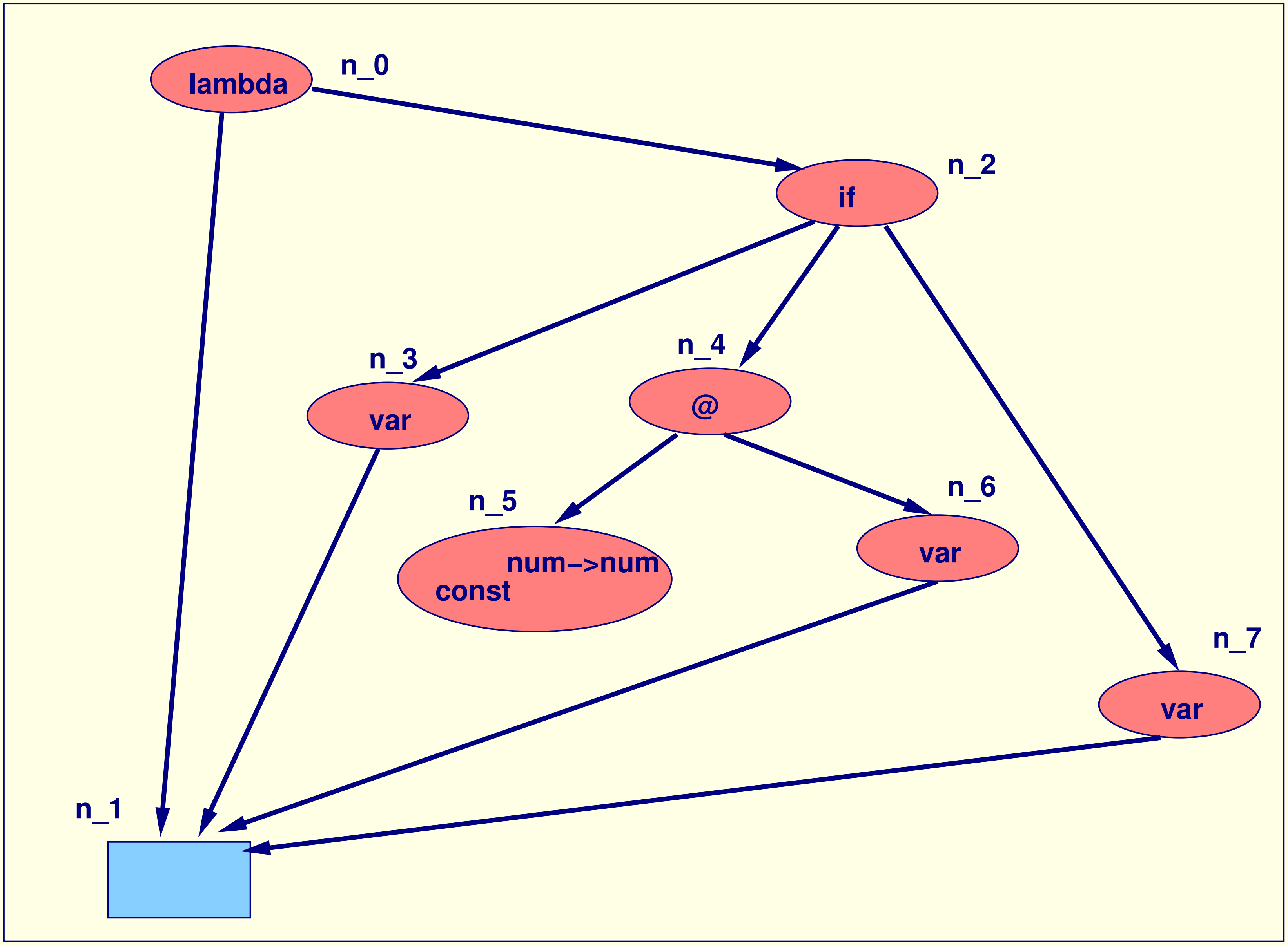
$$b_2 : w' \xrightarrow{f.i} v$$

$$\begin{array}{llll} ((b_1)^{-1}ab_2)^{-1} & \longrightarrow_R & (b_2)^{-1}((b_1)^{-1}a)^{-1} & \notin P^U \\ & \overset{*}{\longrightarrow}_R & (b_2)^{-1}a^{-1}b_1 & \in P^U \end{array}$$

Theorem 1. *(P^U Weak Subject Reduction)*

Let G be a unification graph and let $G \vdash_{P^U} p : u \xrightarrow{l} v$. If p' is the normal form of p under R/A rewriting, then, $G \vdash_{P^U} p' : u \xrightarrow{l} v$.

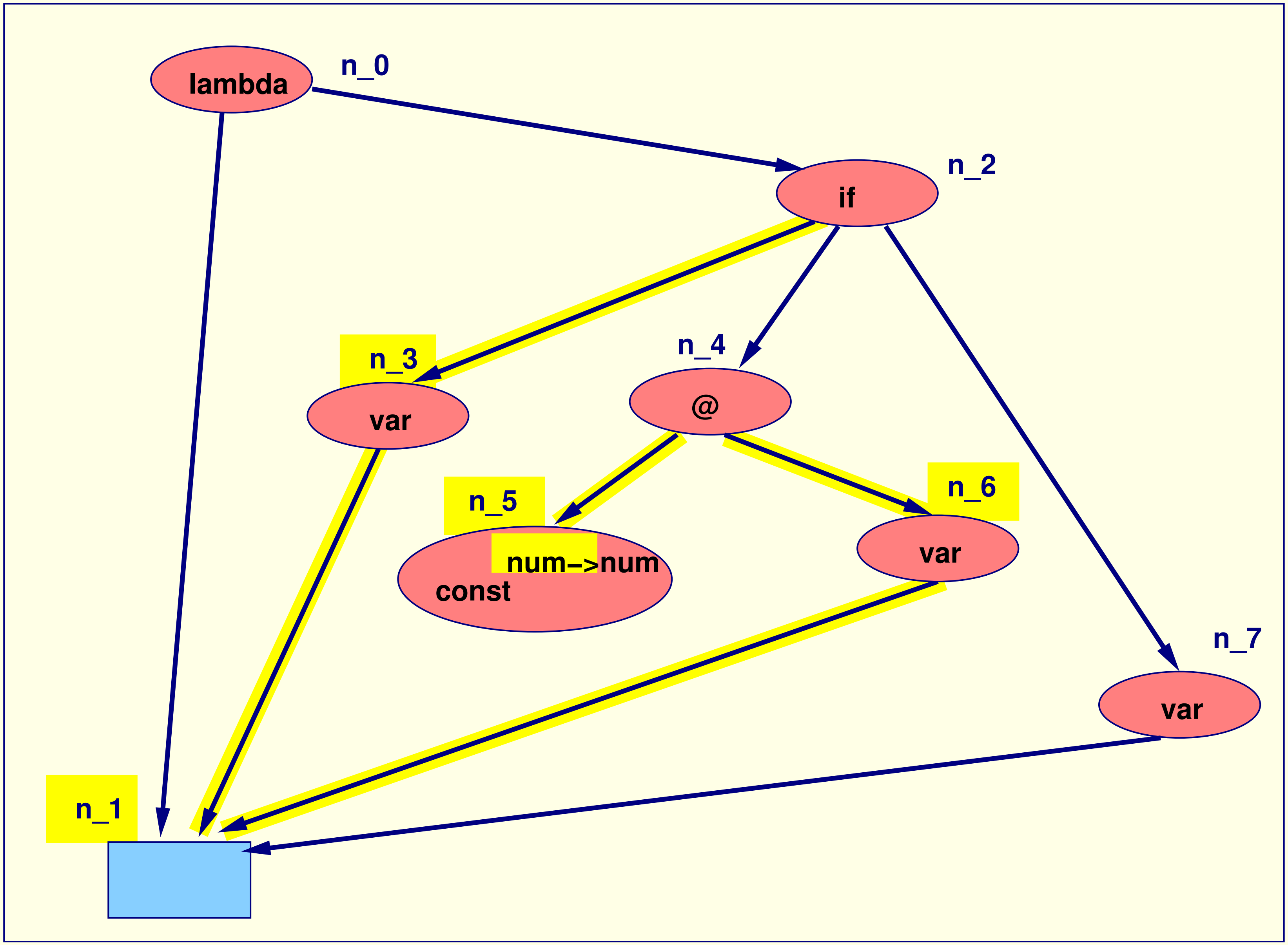
Abstract Syntax Graph



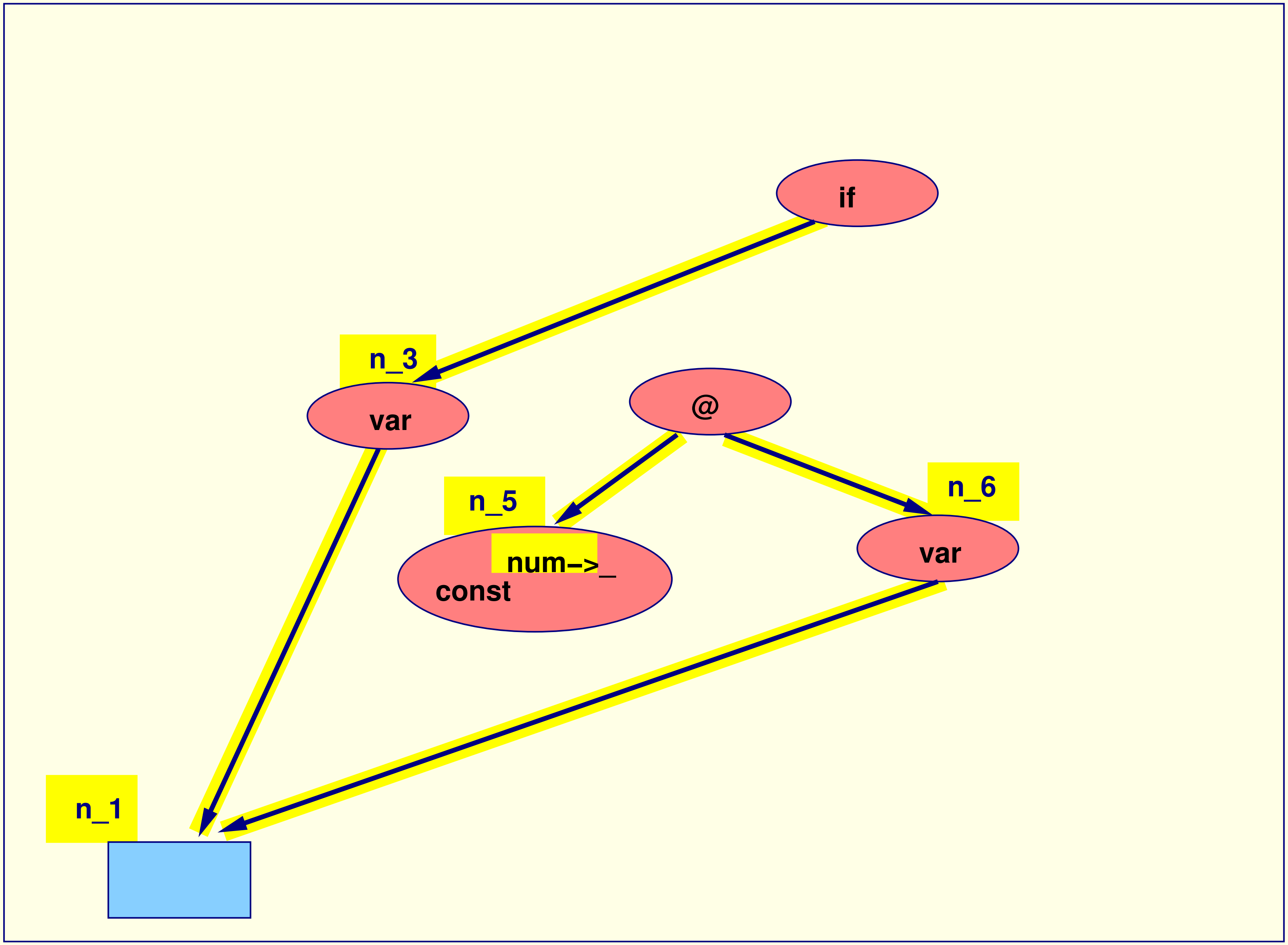
$$\lambda_0\ x_1.\ \mathbf{if}_2\ x_3\ \mathbf{then}\ @_4(\mathit{inc}_5\ x_6)\ \mathbf{else}\ x_7$$

Graph of Program Slice for clash 1

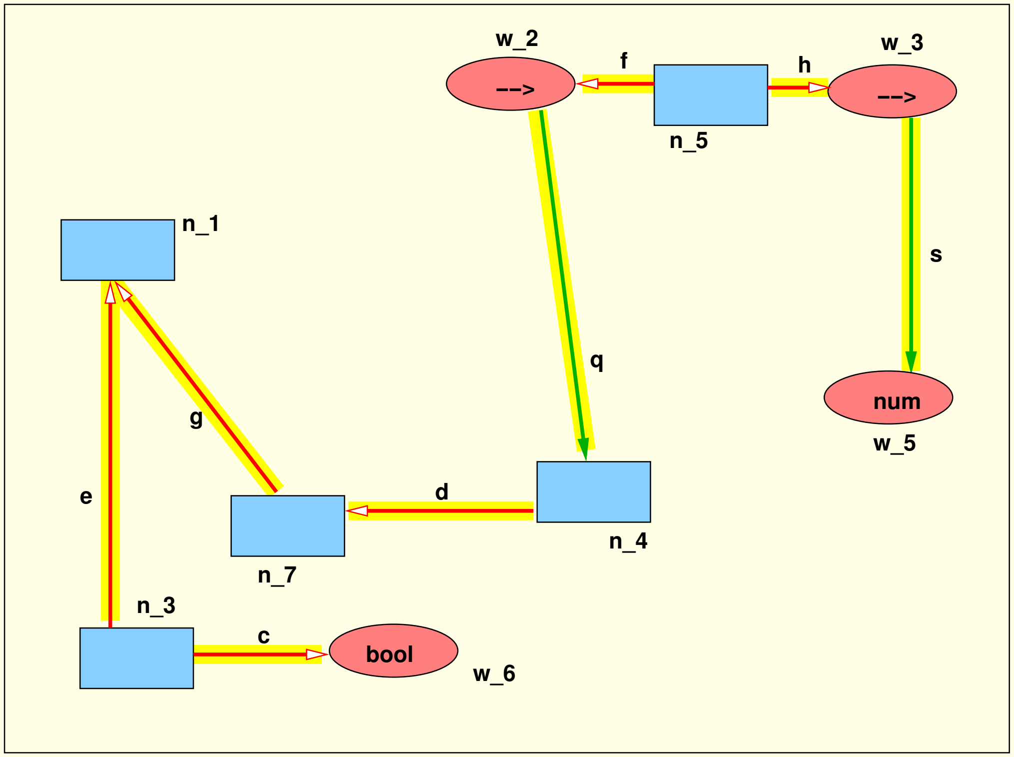
Graph of Program Slice for clash 1



Graph of Program Slice for clash 1



Type Equation Slice for clash 2



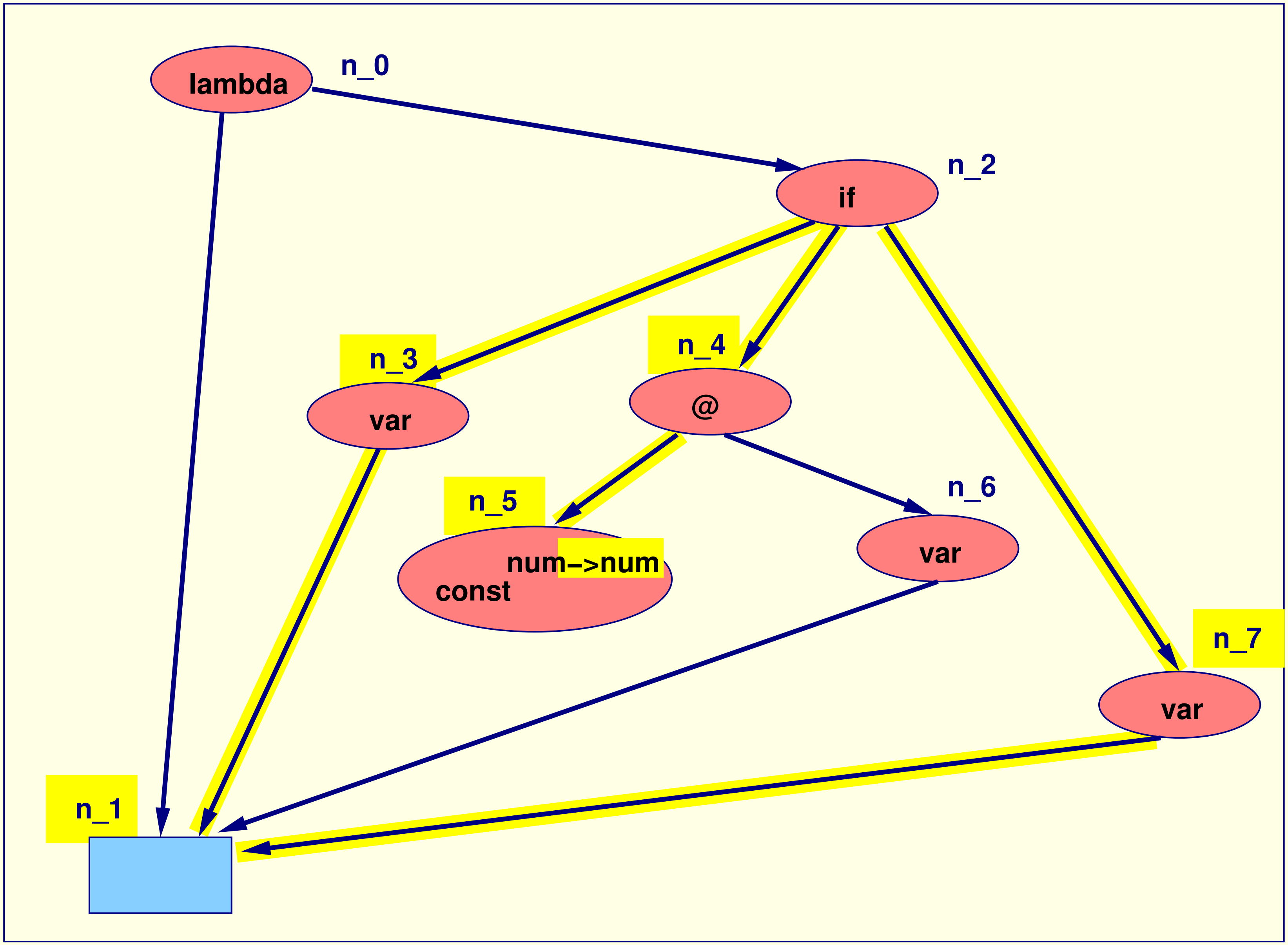
$$\begin{array}{lcl}
\mathsf{bool} & \stackrel{?}{=} & n_3 \\
n_3 & \stackrel{?}{=} & n_1 \\
n_1 & \stackrel{?}{=} & n_7 \\
n_7 & \stackrel{?}{=} & n_4 \\
n_5 & \stackrel{?}{=} & \square{\longrightarrow}n_4 \\
n_5 & \stackrel{?}{=} & \square{\longrightarrow}\mathsf{num}
\end{array}$$

Program Slice S_2

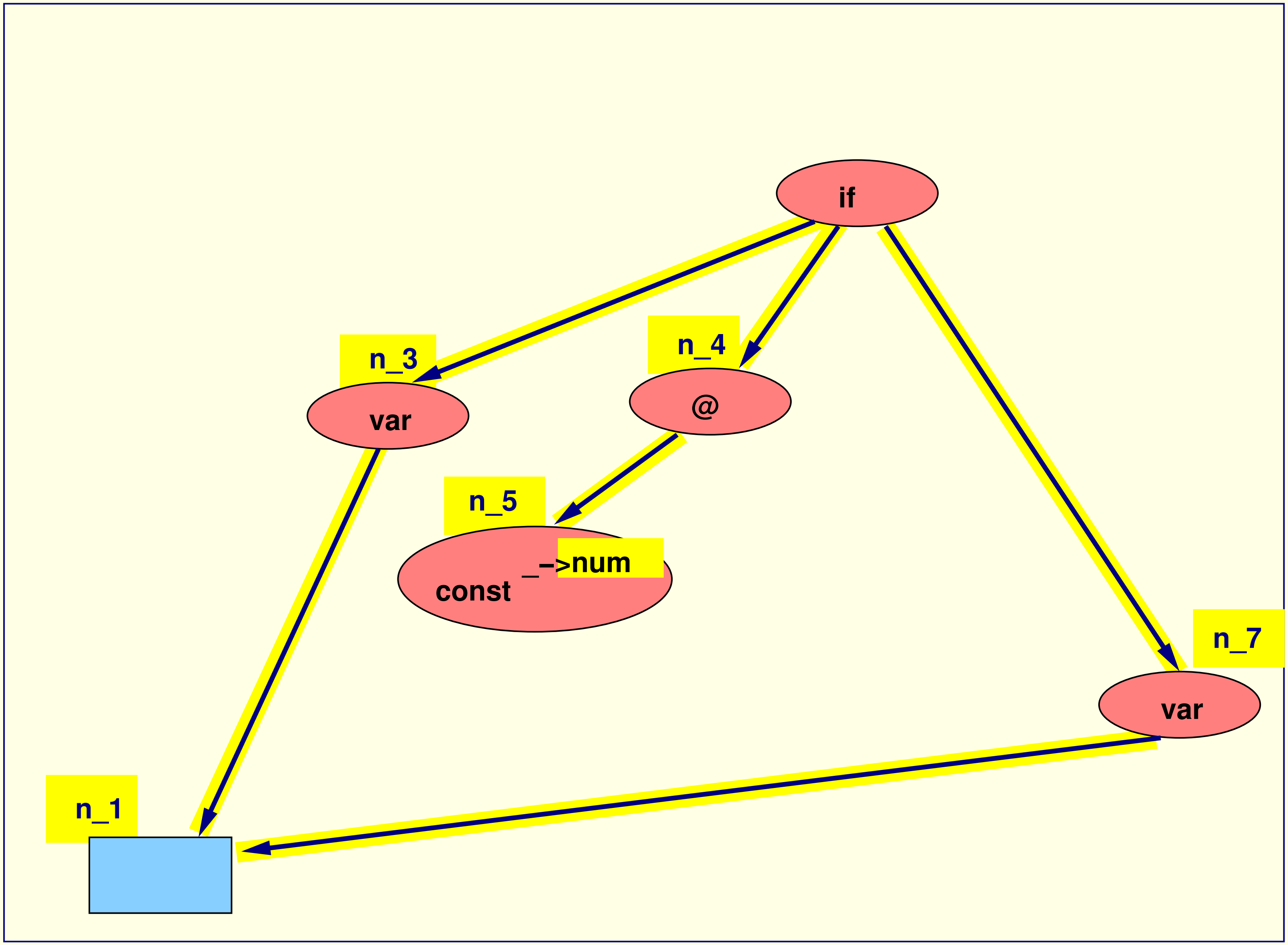
$$
\begin{array}{lclclcl}
\square & = & \mathsf{if}(n_3, \square, \square) & \mapsto & \mathsf{bool} & \stackrel{?}{=} & n_3 \\
n_3 & = & \lambda\mathsf{var}(n_1) & \mapsto & n_3 & \stackrel{?}{=} & n_1 \\
n_7 & = & \lambda\mathsf{var}(n_1) & \mapsto & n_7 & \stackrel{?}{=} & n_1 \\
\square & = & \mathsf{if}(\square, n_4, n_7) & \mapsto & n_4 & \stackrel{?}{=} & n_7 \\
n_4 & = & @(n_5, \square) & \mapsto & n_5 & \stackrel{?}{=} & \square \rightarrow n_4 \\
n_5 & = & \mathsf{const}(\square \rightarrow \mathit{num}) & \mapsto & n_5 & \stackrel{?}{=} & \square \rightarrow \mathit{num}
\end{array}
$$

Graph of Program Slice S_2

Graph of Program Slice S_2



Graph of Program Slice S_2



Related Research

Reason lists [Wand '86]

- Accumulate reason lists during unification
- Lacks soundness: not enough reasons accumulated to simulate error
- No way of eliminating unwanted reasons: lacks cancellative rules

Flow techniques [Johnson-Walz'86]

- Error-tolerant unification
- Complicated algorithm, informally stated

Explanation-based systems [Stansifer '94, Duggan '94, Soosaipillai '90]

- Interactive graph navigation
- Lack automation

Others Approaches:

- Automata-based approach [Gandhe et al. '96]
- TCC explanation in PVS [SRI, '98]

Logic Programming

Unification failure [Cox, '87, Port, '88]
Maximally unifiable subsets [Cox,'84, Chen et al.'86]
Tracing [Ducassé, '99]
Visual Debuggers [Deransart, '2000]

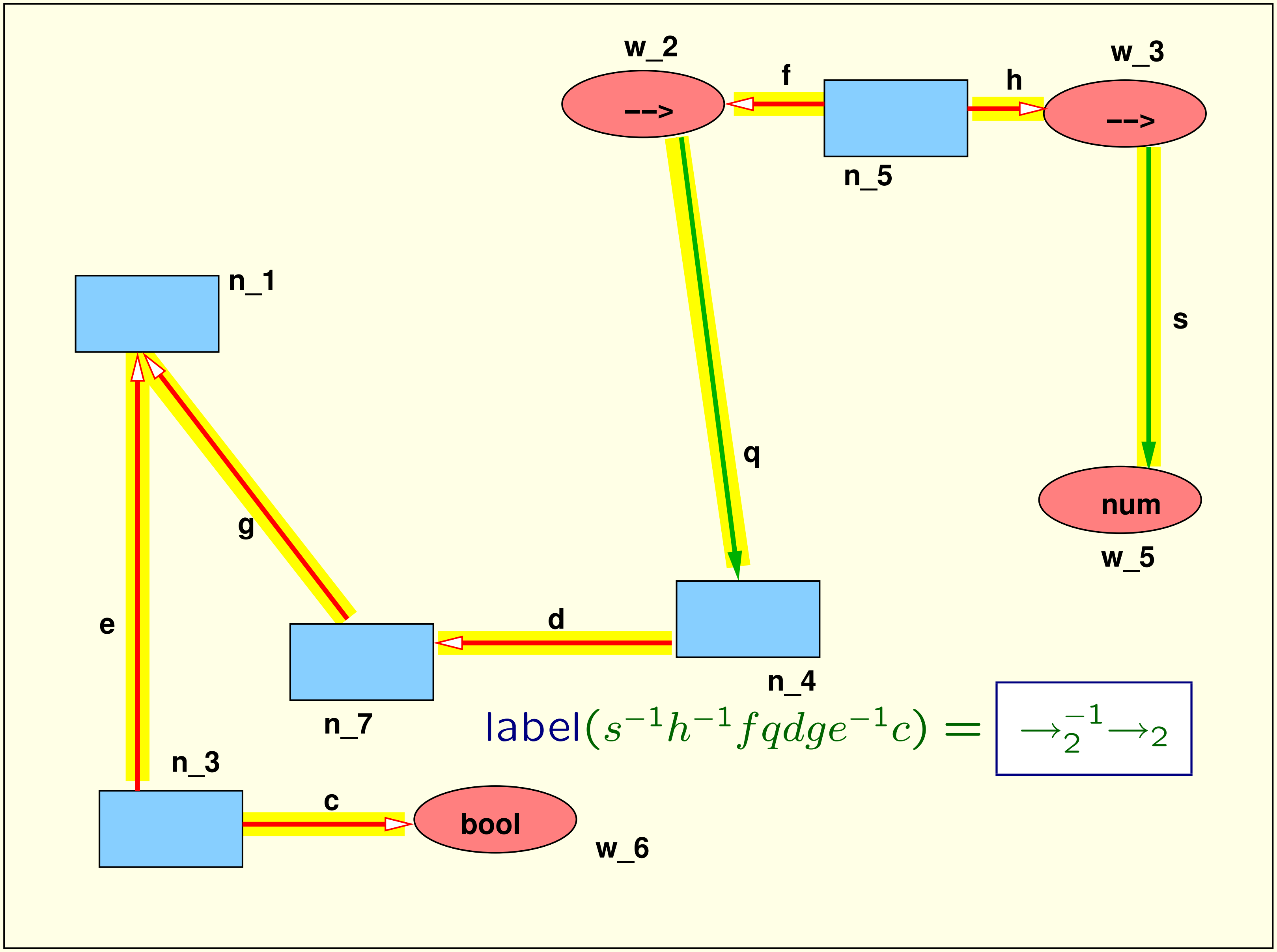
Rewrite Systems

Origin tracking [Bertot, '95, van Deursen et al. 93]

Artificial Intelligence

conflict sets [Reiter '87, de Kleer '92]
Explanation-based diagnosis [Genesereth '84, Wick and Thompson '92]

Clash 2



$$\text{label}(s^{-1}h^{-1}fqdge^{-1}c) = \boxed{\longrightarrow_2^{-1}\longrightarrow_2}$$

Unification Logic LE_0 of Le Chenadec

$$s \qquad \frac{M = N}{N = M}$$

$$t \qquad \frac{M = x \qquad x = N}{M = N}$$

$$s_i \qquad \frac{f(M_1, M_2) = f(N_1, N_2)}{M_i = N_i} \qquad\qquad i \in \{1, 2\}$$

$$su \qquad \frac{x = M \qquad y = C[x]}{y = C[M]}$$

LE_0 is sound and complete with respect to path connectivity in the quotient graph [Proposition 2.10, Le Chenadec '89]

P^U and LE_0 are equivalent because they are sound and complete with respect to the same model.

P^U works on vertices of a labeled directed graph, making apparent the integration with unification algorithms.

Geometric interpretation of proofs due to Le Chenadec. But connection with semi-Dyck sets provides opportunity to apply algorithms for formal language path problems to unification source tracking.