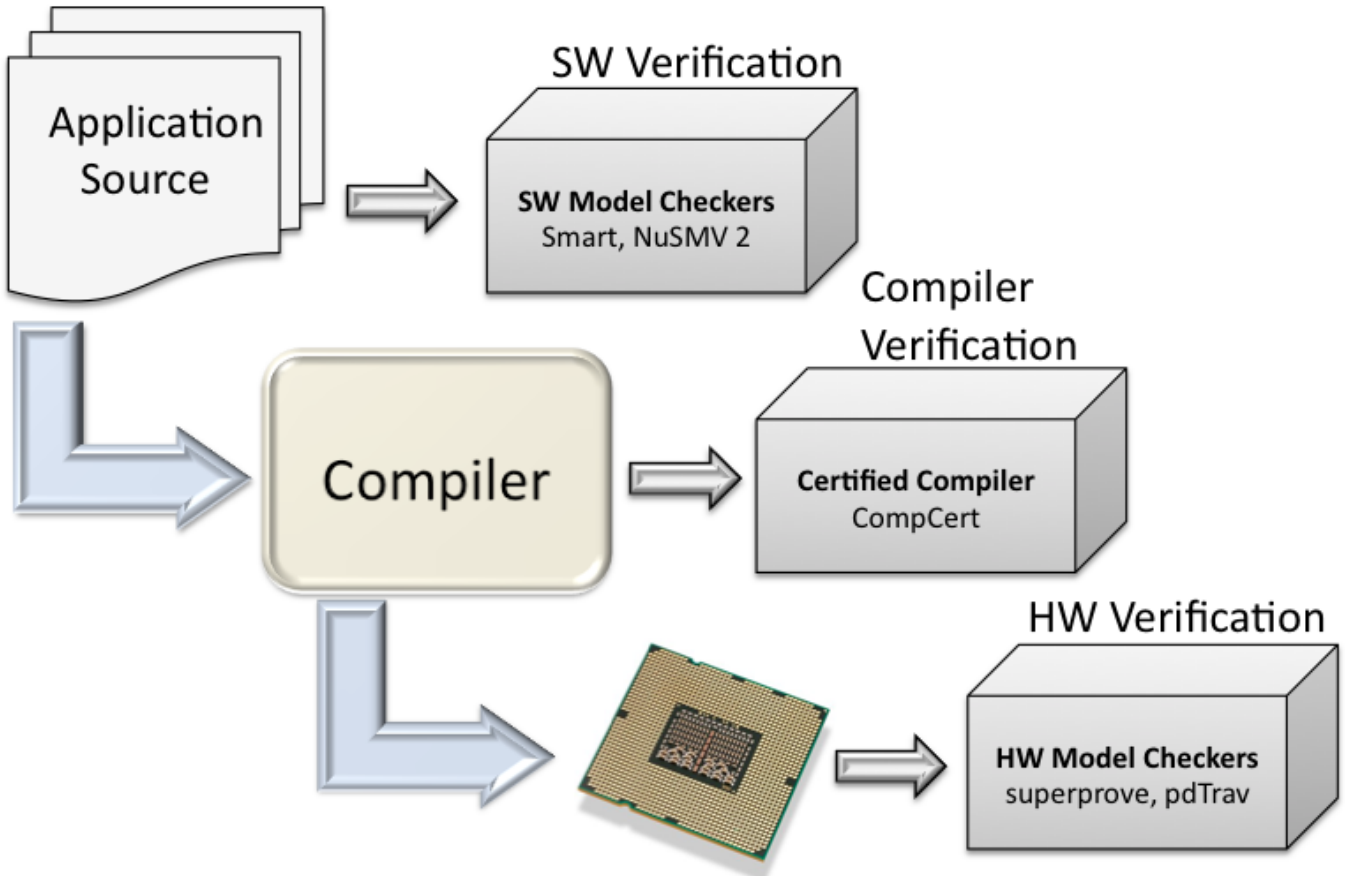
# Who Watches the Watchers: Toward Provably-correct Decision Diagram Code

**Yousra Lembachar**, Ryan Rusich,
Iulian Neamtiu, Gianfranco Ciardo

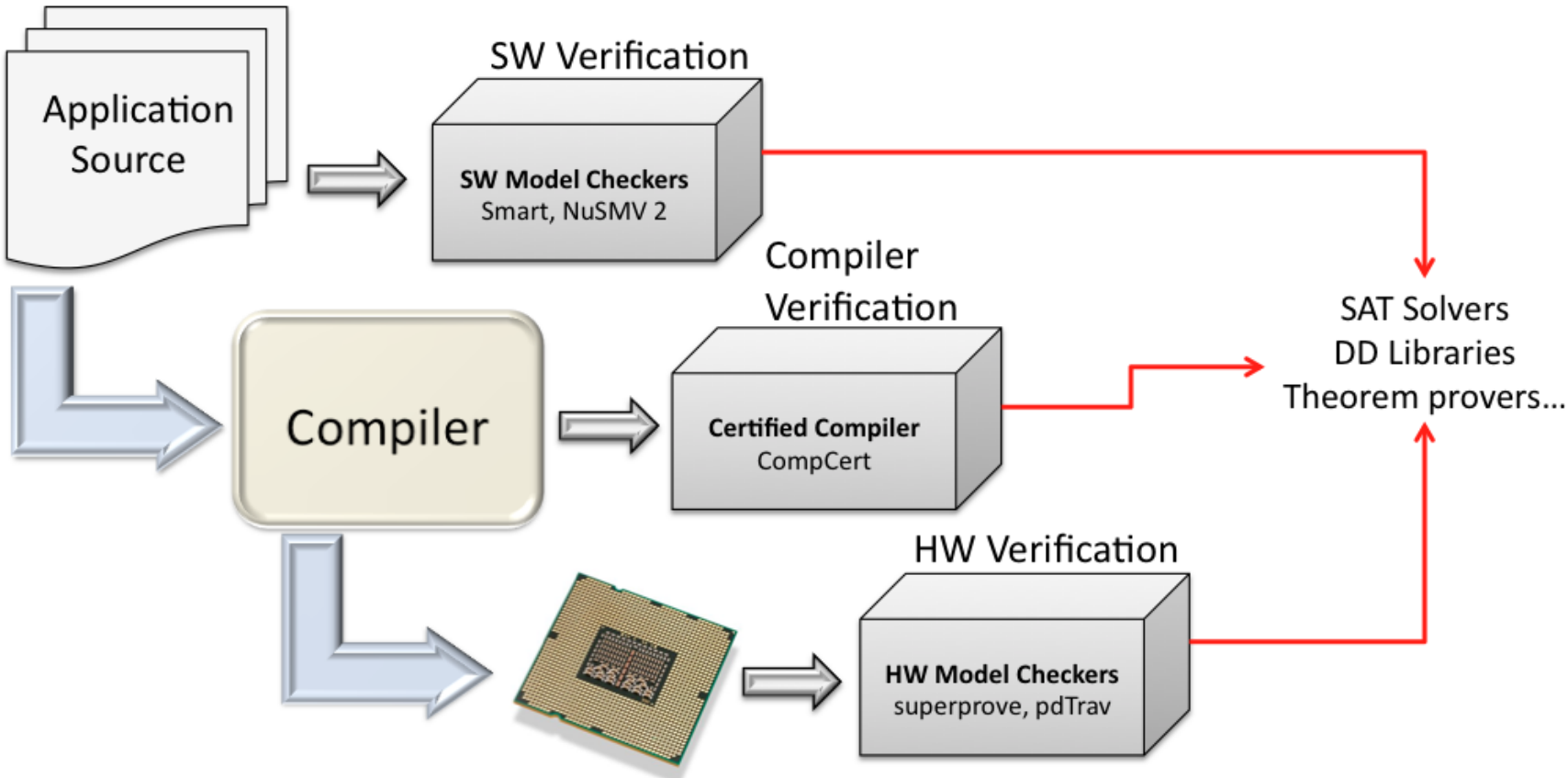
University of California, Riverside

# Toward a Completely Verified Software Toolchain

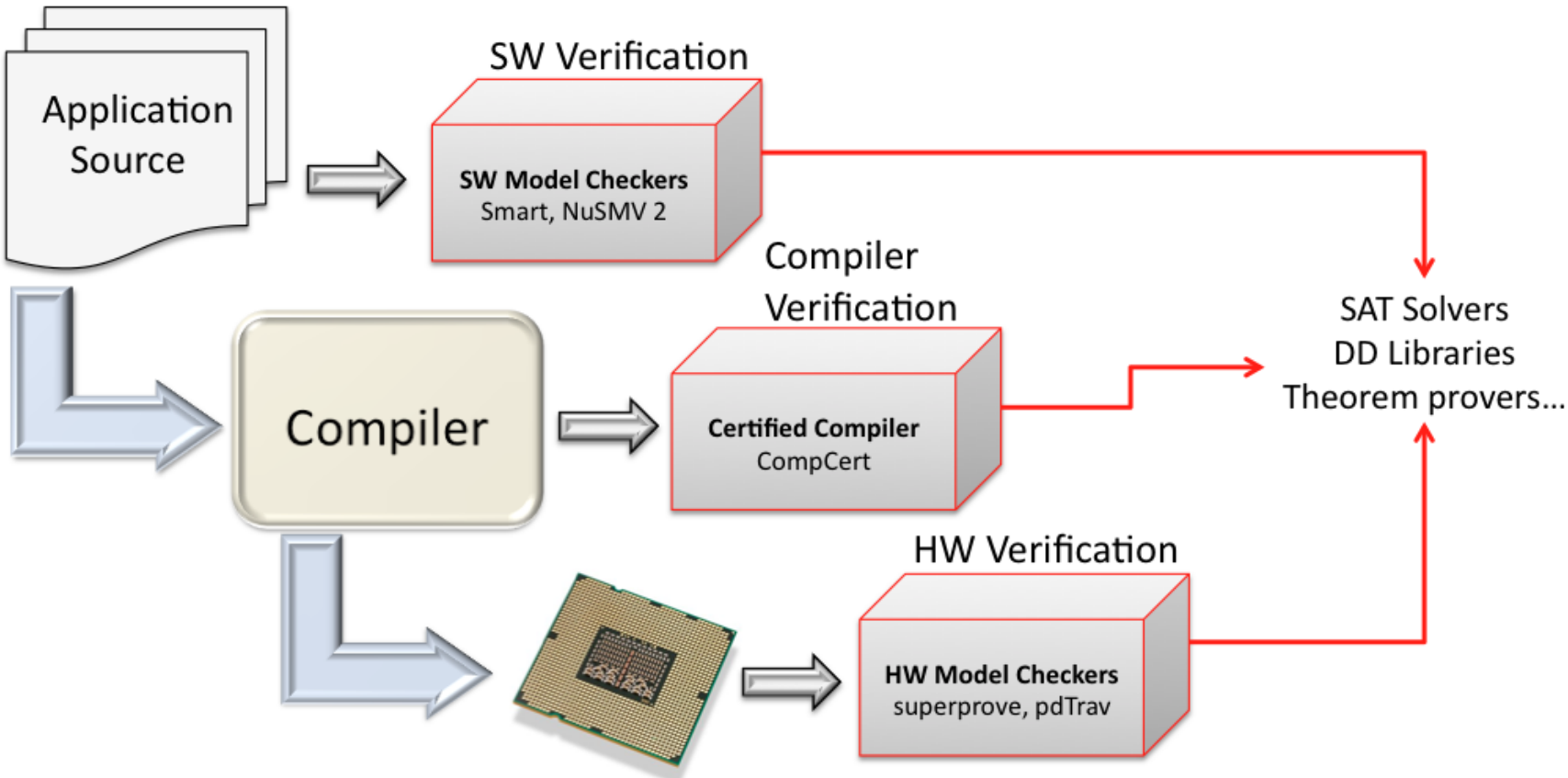



Trusted tool-chain!

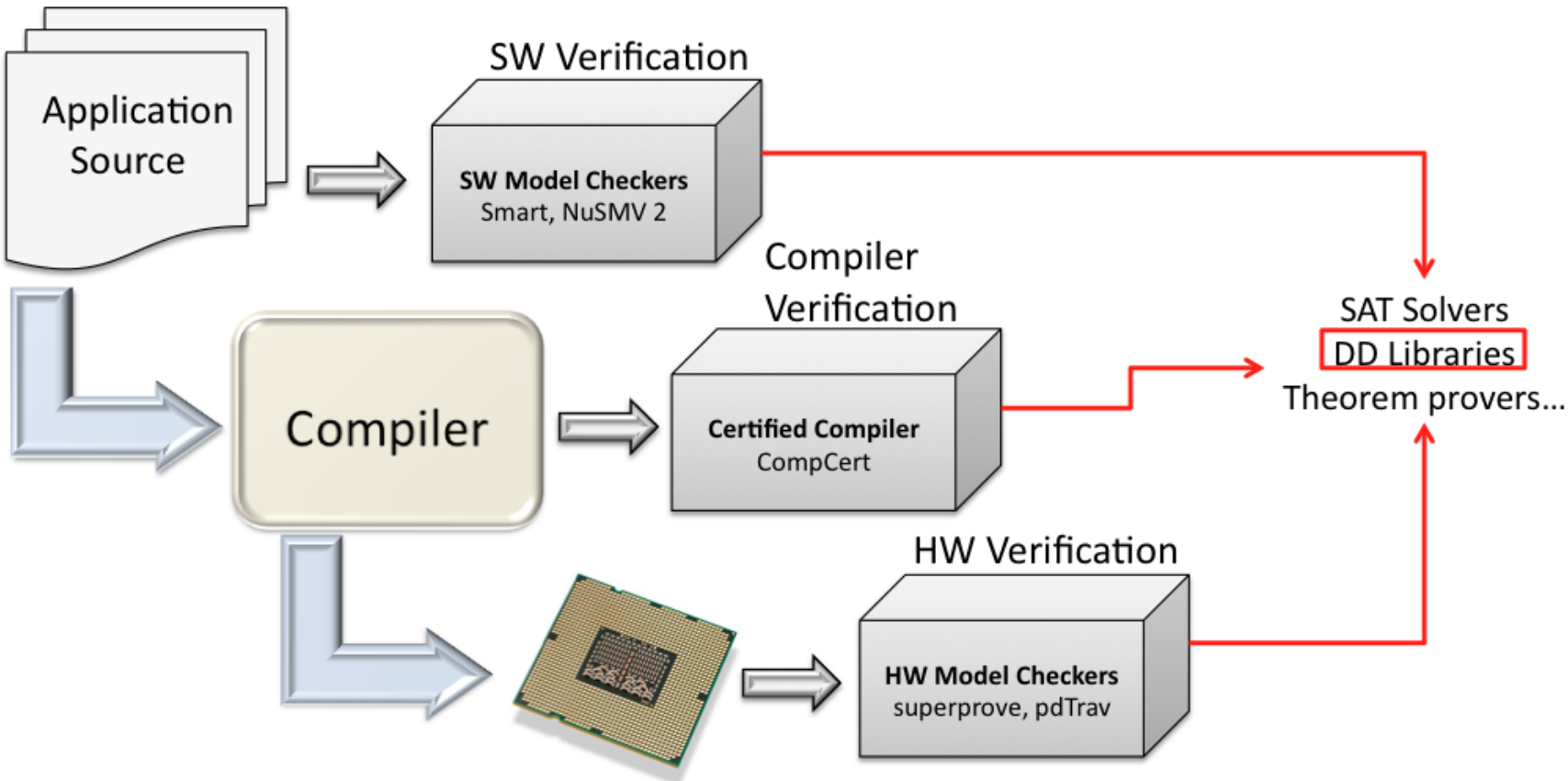
# Toward a Completely Verified Software Toolchain



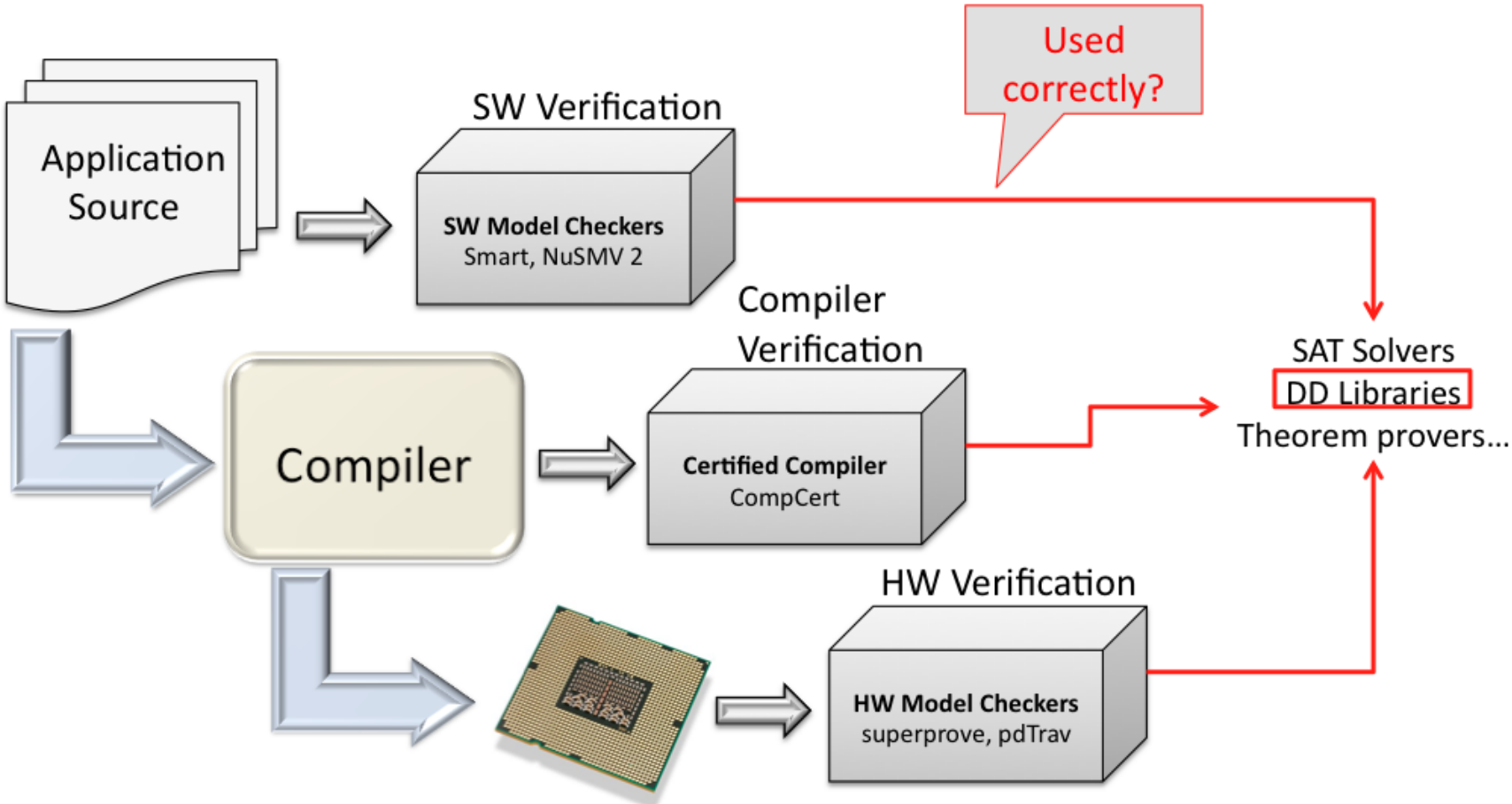
# Toward a Completely Verified Software Toolchain



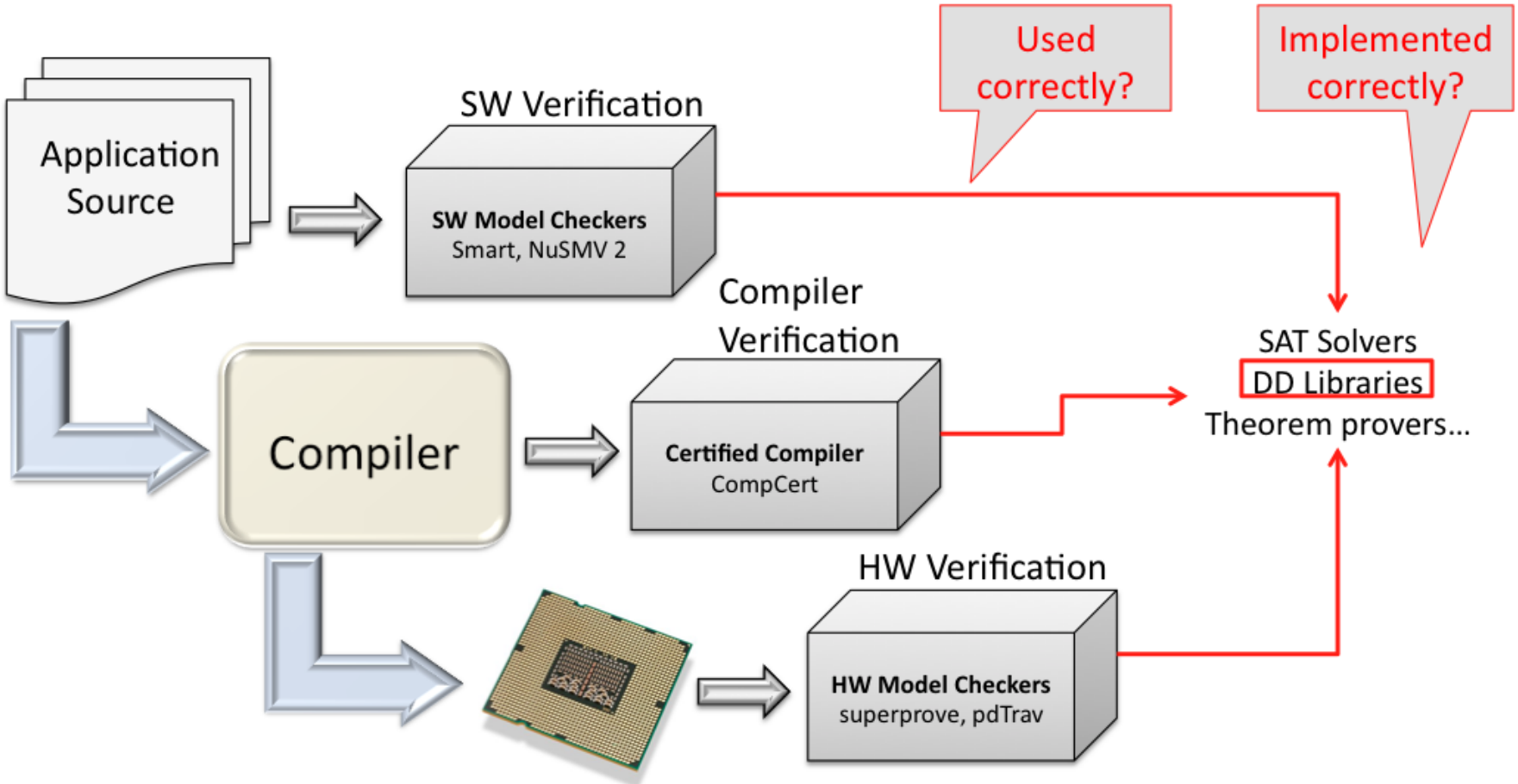
# Toward a Completely Verified Software Toolchain



# Toward a Completely Verified Software Toolchain



# Toward a Completely Verified Software Toolchain

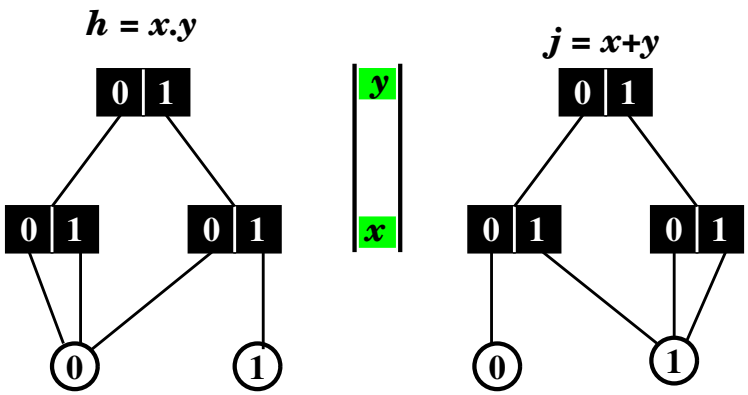


# Our Contribution

- BDDL
  - A calculus for reasoning about decision diagram library **and** client code
  - Provides a sound type system with operational semantics
  - Enables compile-time detection of dynamic errors
  - Enforces correct structural properties and semantics for decision diagrams
- Demonstrate the efficacy of our approach via real world bugs detected in three mature libraries: CUDD (NuSMV), MDDL (SMART), and JavaBDD

# Runtime Error Example in CUDD

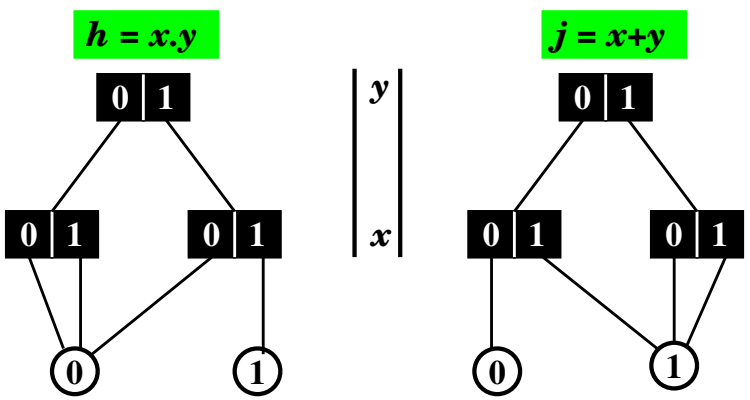
```
int main () {
  Cudd mgr(0,2);
  BDD x = mgr.bddVar();
  BDD y = mgr.bddVar();
  BDD h = x * y;
  BDD j = x + y;
  BDD k = h.Compose(j,2);
}
```




```
DdNode* Cudd_bddCompose(
  DdManager * dd, DdNode * f,
  DdNode * g, int v) {
    DdNode *proj, *res;
    /* Sanity check.  */
    if (v < 0 || v >= dd->size)
    return(NULL);
    proj =dd->vars[v];
    do {
    ...
    } while (dd->reordered == 1);
     return(res);}
     BDD BDD::Compose(BDD g, int v)
    { ...
     return BDD(...,
     Cudd_bddCompose(
     mgr.node, g.node, v));
    }
```

# Runtime Error Example in CUDD

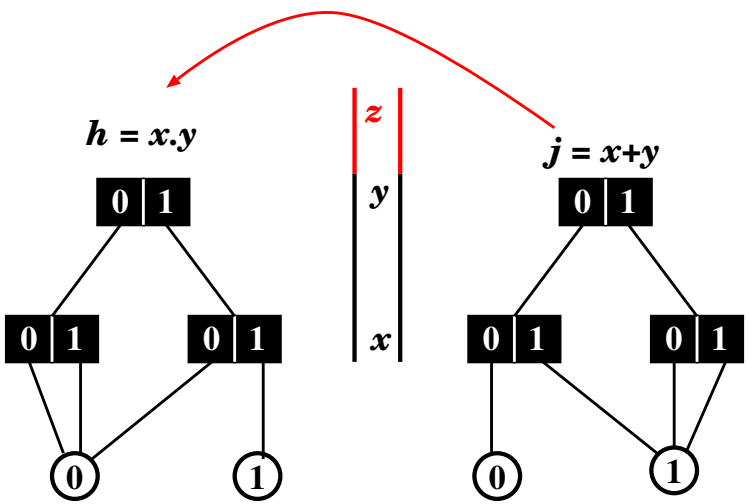
```
int main () {
  Cudd mgr(0,2);
  BDD x = mgr.bddVar();
  BDD y = mgr.bddVar();
  BDD h = x * y;
  BDD j = x + y;
  BDD k = h.Compose(j,2);
}
```




```
DdNode* Cudd_bddCompose(
  DdManager * dd, DdNode * f,
  DdNode * g, int v) {
    DdNode *proj, *res;
    /* Sanity check.  */
    if (v < 0 || v >= dd->size)
    return(NULL);
    proj =dd->vars[v];
    do {
    ...
    } while (dd->reordered == 1);
     return(res);}
     BDD BDD::Compose(BDD g, int v)
    { ...
     return BDD(...,
     Cudd_bddCompose(
     mgr.node, g.node, v));
    }
```

# Runtime Error Example in CUDD

```
int main () {
  Cudd mgr(0,2);
  BDD x = mgr.bddVar();
  BDD y = mgr.bddVar();
  BDD h = x * y;
  BDD j = x + y;
  BDD k = h.Compose(j,2);
}
```

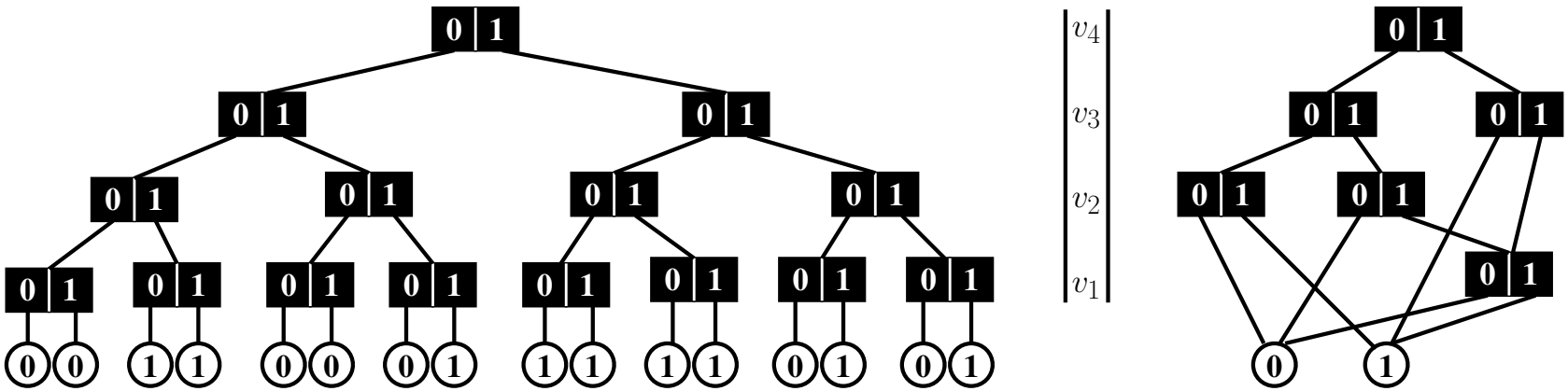



```
DdNode* Cudd_bddCompose(
  DdManager * dd, DdNode * f,
  DdNode * g, int v) {
    DdNode *proj, *res;
    /* Sanity check.  */
    if (v < 0 || v >= dd->size)
    return(NULL);
    proj = dd->vars[v];
    do {
    ...
    } while (dd->reordered == 1);
    return(res);}
     BDD BDD::Compose(BDD g, int v)
    { ...
      return BDD(...,
      Cudd_bddCompose(
      mgr.node, g.node, v));
    }
```

# Binary Decision Diagrams (BDDs)

Binary tree

BDD

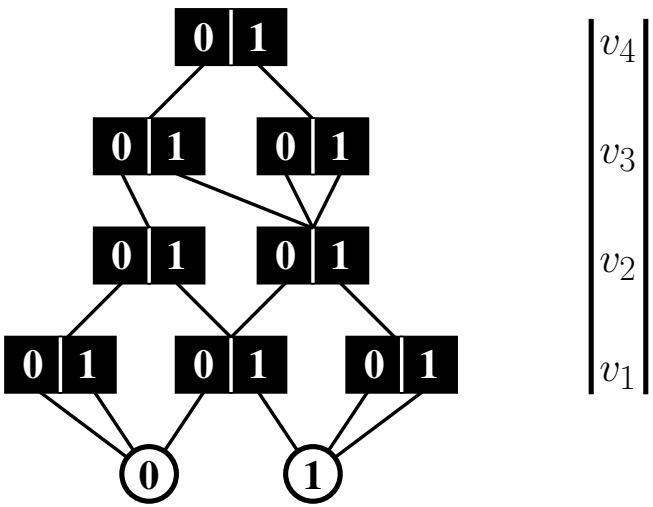



$$f(v_1, v_2, v_3, v_4) = (v_4 \vee v_2) \wedge (v_3 \rightarrow v_1)$$

# BDD Encodings - Sets and n-ary relations

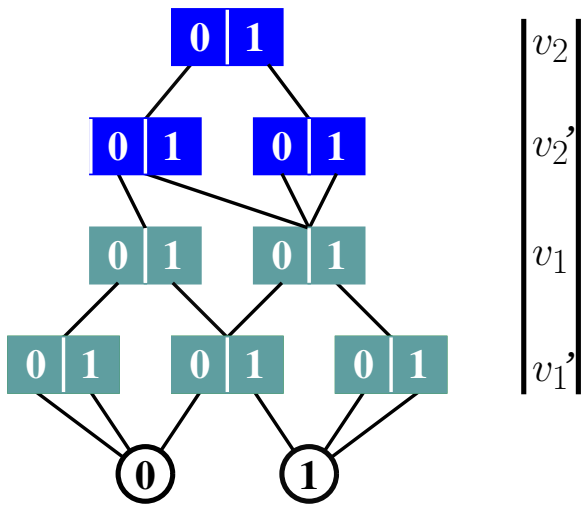
## Set

{0011, 0101, 0110, 0111, 1001,
1010, 1011, 1101, 1110, 1111}




## Binary relation

{(01,01) (00,11) (01,10) (01,11)
(10,01) (11,00) (11,01) (10,11)
(11,10) (11,11)}

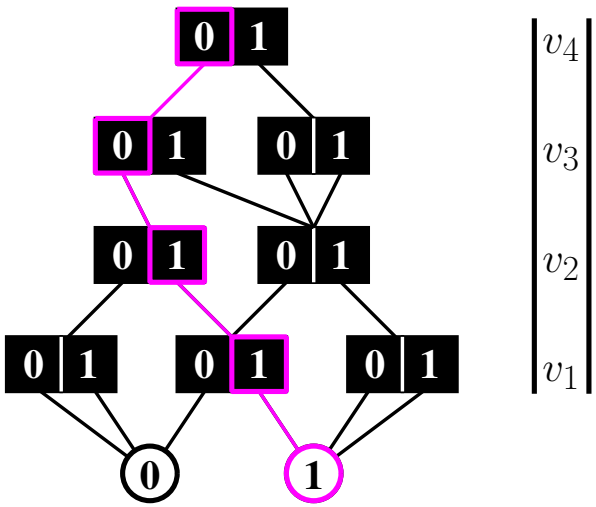


# BDD Encodings - Sets and n-ary relations

## Set

$v_4 v_3 v_2 v_1$

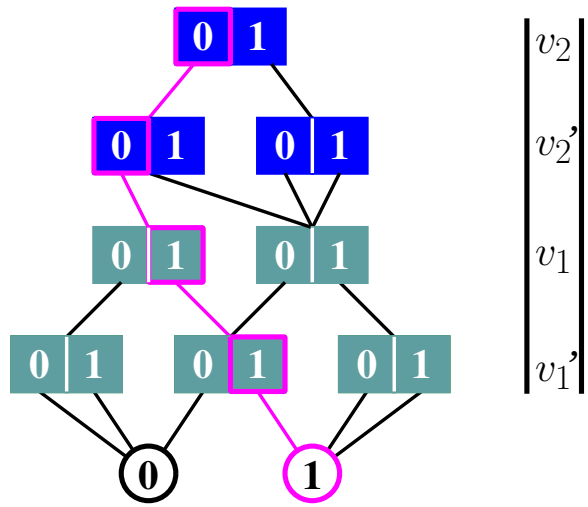
(0 0 1 1)




## Binary relation

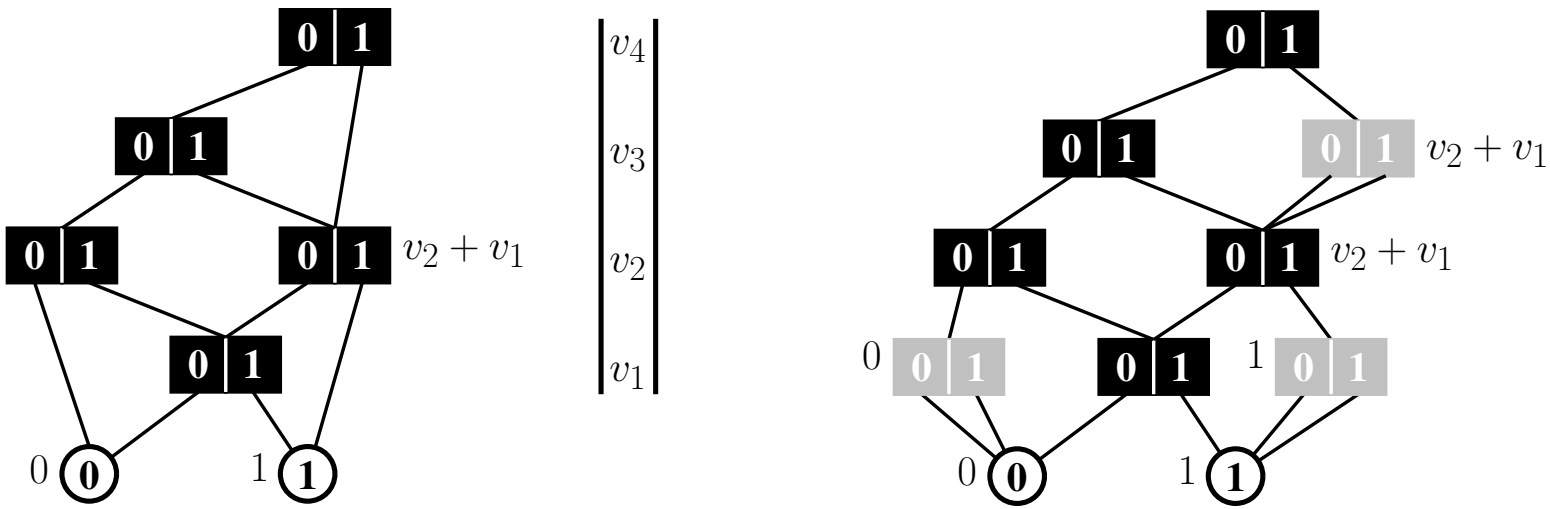
$v_2\ v_1\ v_2'\ v_1'$

(0 1, 0 1)



# BDD Reduction Rules

Reduction rules for canonicity and compactness...




Fully-reduced (left) vs. quasi-reduced (right) BDDs

# BDDL Calculus

Terms

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```

Types

$$
\begin{array}{l}
\tau ::= \\
\quad \texttt{bool} \mid \texttt{nat} \\
\mid \texttt{string} \mid \texttt{Id} \\
\mid \tau \to \tau \\
\mid \texttt{ref}\ \tau \\
\mid \texttt{bdd[l, r, c]} \\
\mid \{\nu : \tau \mid p(\pi)\}
\end{array}
$$

Typing rules (example)

$$
\frac{\begin{array}{c}
\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : string \\
\Gamma \vdash \mathtt{t}_0 : \{\nu : nat \mid \nu \geq 1 \wedge \nu = l\} \\
\Gamma \vdash \mathtt{t}_1 : \ ref\ \ bdd[l', r, c] \\
bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\
\Gamma \vdash \mathtt{t}_2 : \ ref\ \ bdd[l'', r, c] \\
bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}
\end{array}}{\Gamma, id : Id \vdash \mathtt{Bnode}\ (\mathtt{t}_0, id, v_{var}, \mathtt{t}_1, \mathtt{t}_2) : bdd[l, r, c]}\ \text{(T-BNODE)}
$$

Operational semantics (example)

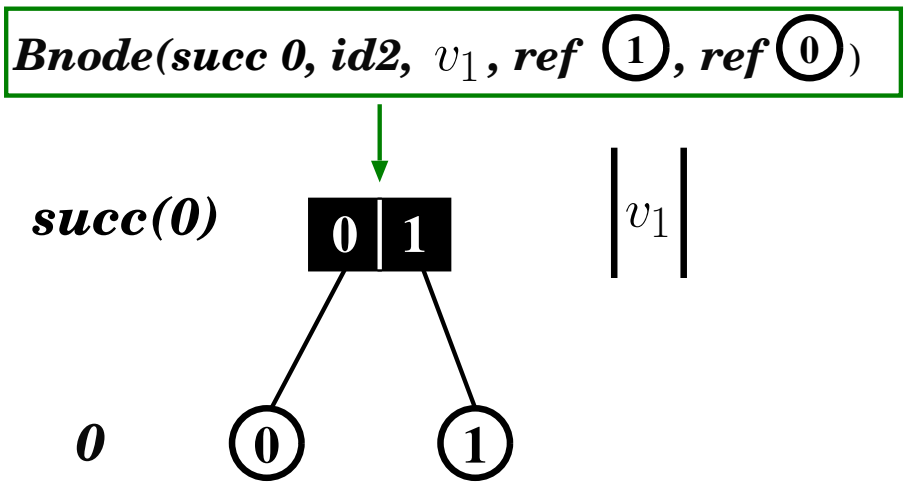
$$
\frac{\mu; \mathtt{t}_j \longrightarrow \mu'; \mathtt{t'}_j}{\begin{array}{l}\mu; \mathtt{Bnode}(\mathtt{v}_i^{i \in 1,..,j-1}; \mathtt{t}_j; \mathtt{t}_k^{k \in j+1,..,5}) \\ \longrightarrow \mu'; \mathtt{Bnode}(\mathtt{v}_i^{i \in 1,..,j-1}; \mathtt{t'}_j; \mathtt{t}_k^{k \in j+1,..,5})\end{array}}\ \text{(E-BNODE)}
$$

Soundness proof

A well-typed BDD program can't go wrong.

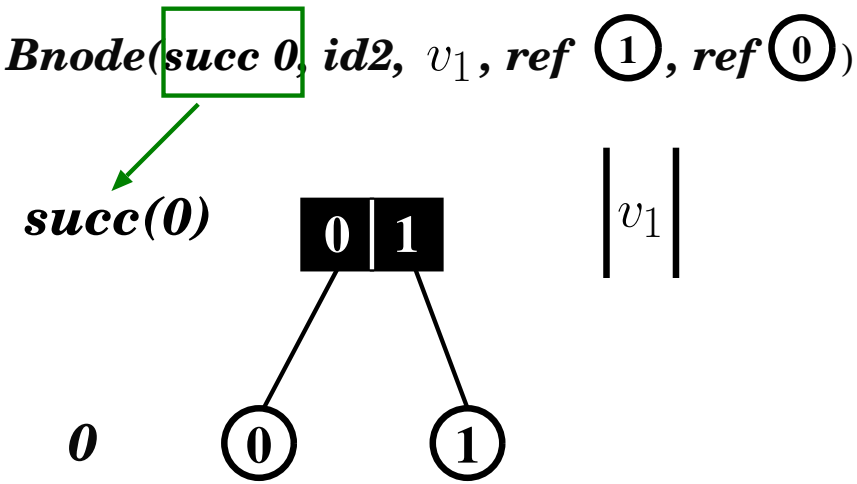
# BDDL Terms - The Bnode term and its attributes

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```



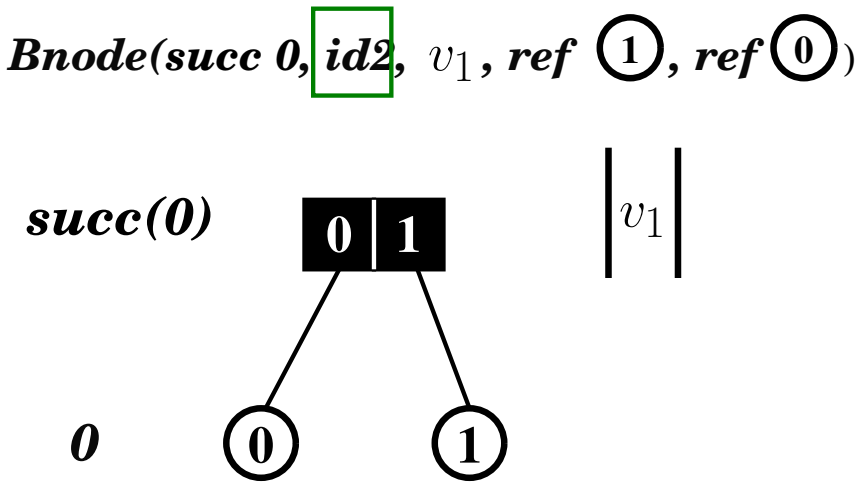
# BDDL Terms - The Bnode term and its attributes

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level  | t.index
 | t.var  | t.tchild
 | t.fchild
```



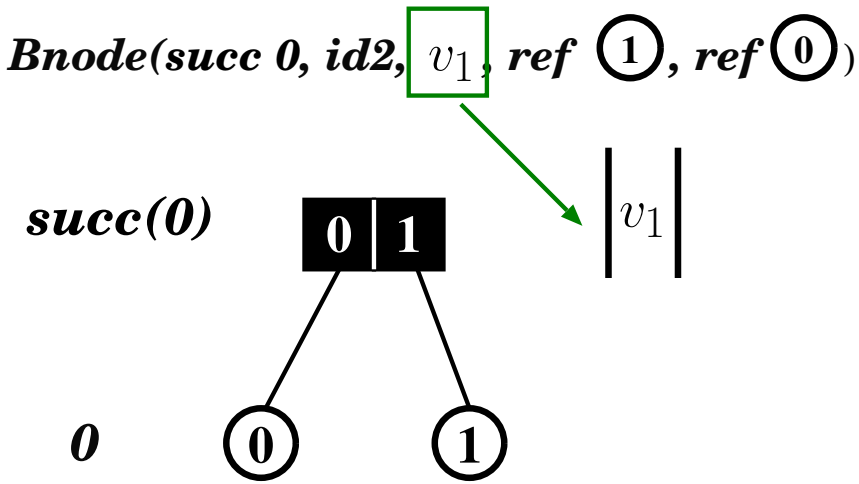
# BDDL Terms - The Bnode term and its attributes

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```



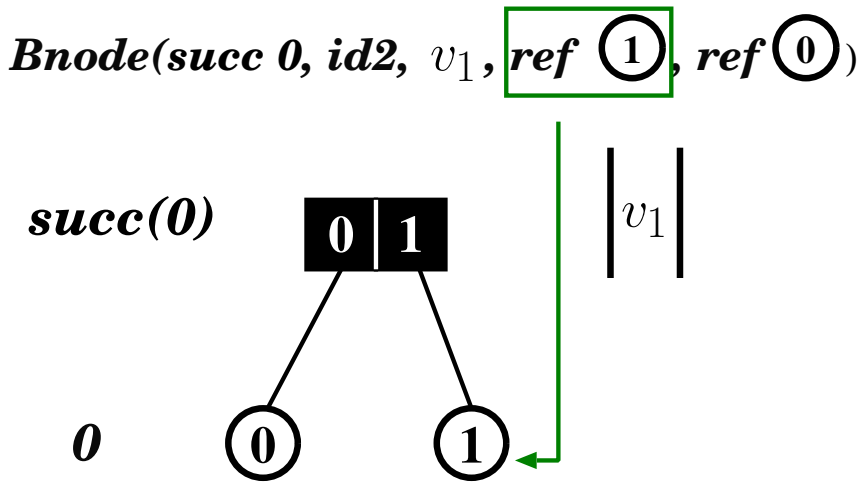
# BDDL Terms - The Bnode term and its attributes

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```



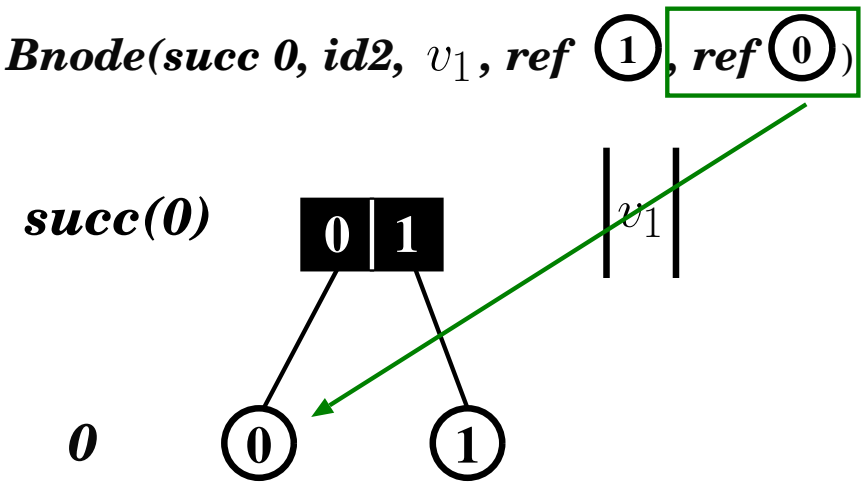
# BDDL Terms - The Bnode term and its attributes

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```



# BDDL Terms - The Bnode term and its attributes

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```

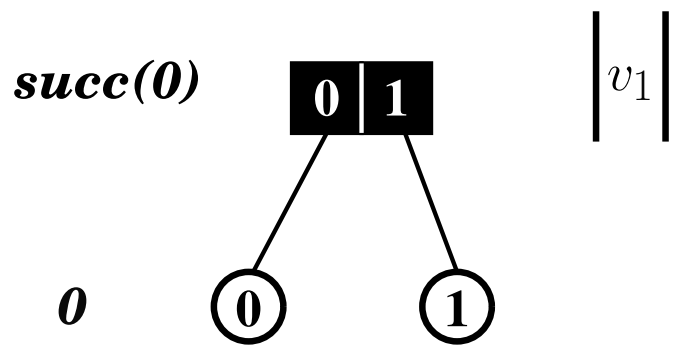


# BDDL Terms - $\lambda$-calculus based terms

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```

```
letrec v1 = ... in
letrec id2 = ... in
letrec build =
λ n .      /* n = succ(0)*/
 λ b .     /* b = true*/
  if iszero n then
     (if b then 1 else 0)
   else
     Bnode (succ(0), id2, v1,
      build 0 true, build 0 false)

   in build (succ(0) true)
```

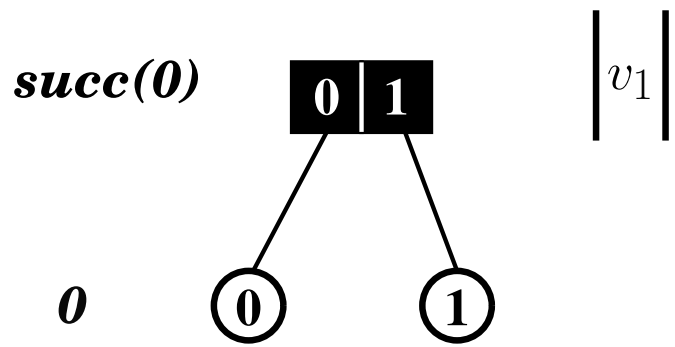


# BDDL Terms - $\lambda$-calculus based terms

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```

```
letrec v1 = ... in
letrec id2 = ... in
letrecbuild =
λ n.     /* n = succ(0)*/
 λ b.    /* b = true*/
  if iszero n then
     (if b then 1 else 0)
   else
     Bnode (succ(0), id2, v1,
      build 0 true, build 0 false)

    in build (succ(0) true)
```

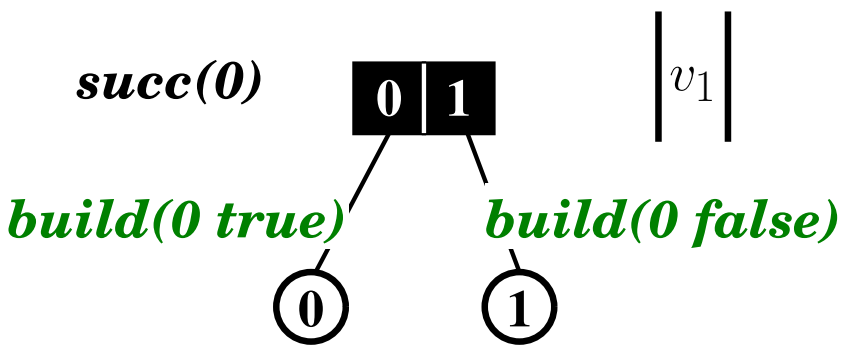


# BDDL Terms - $\lambda$-calculus based terms

```
t ::=
  v | x
 | succ t | pred t
 | iszero t
 | λ:τ.t
 |  t t
 | letrec x : τ = t in t
 | if then t else t t
 | ref t | !t
 | Bnode (t, i, t, t, t)
 | t.level | t.index
 | t.var | t.tchild
 | t.fchild
```

```
letrec v1 = ... in
letrec id2 = ... in
letrecbuild =
λ n.
 λ b.
  if iszero n then
     (if b then 1 else 0)
   else
     Bnode (succ(0), id2, v1,
      build 0 true, build 0 false)

     in build (succ(0) true)
```



# BDDL Types

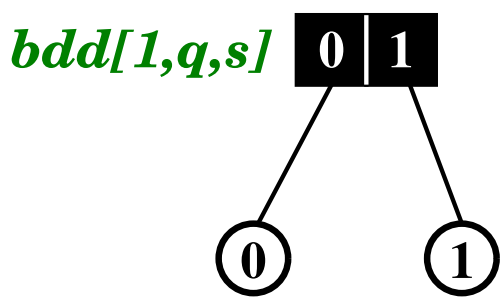
$\tau$ :=
  bool | nat
  | string | Id
  | $\tau \to \tau$
  | ref $\tau$
  | bdd[l,r,c]
  | $\nu : \tau$ | $p(\pi)$

l := $\bot$, $nv$
r := $\bot$, f, q
c := $\bot$, s, e

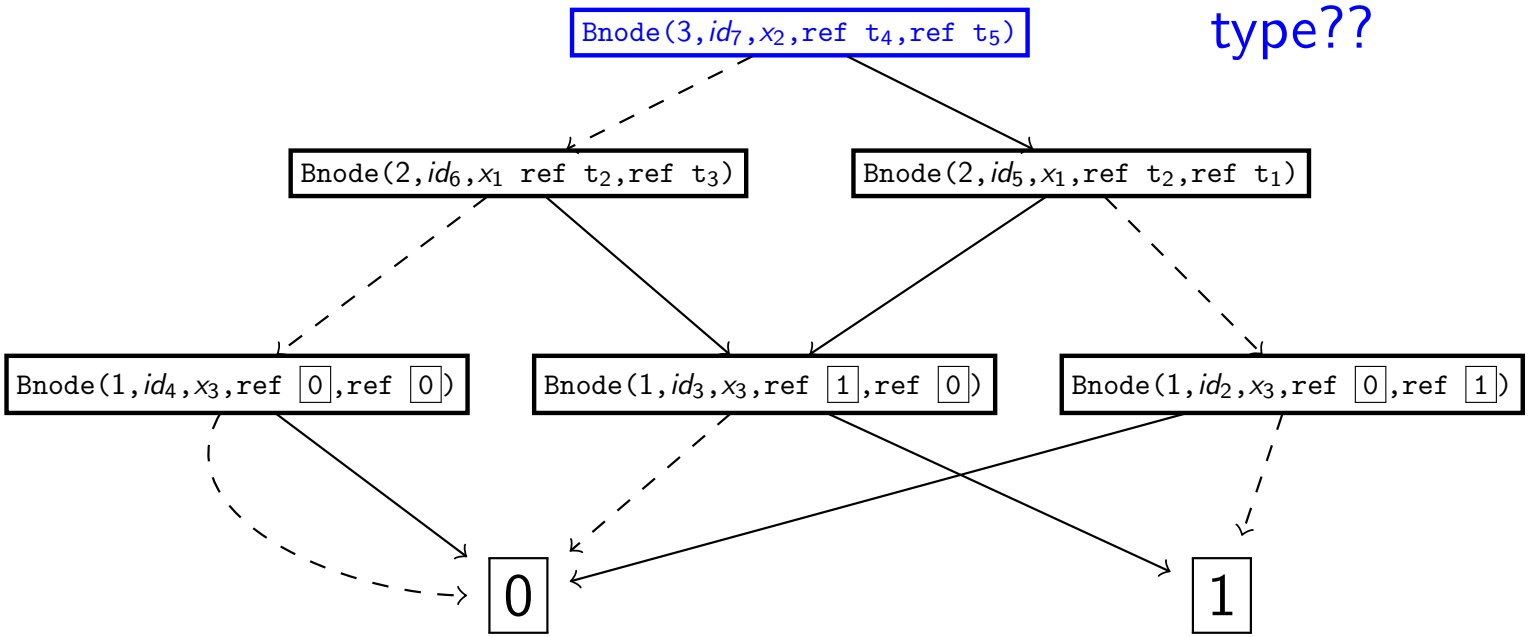
$\pi$ := $\nu \mid l \mid r \mid c$

letrec $v_1$ : string = ... in
letrec $id_2$ : Id = ... in
letrec build :
$\{\nu : nat \mid \nu = l\} \to bool \to bdd[l, r, c] \mid l \leq 1$
$\lambda$ n .
 $\lambda$ b .
 if iszero n then
    (if b then 1 else 0)
  else
  Bnode (succ(0), $id_2$, $v_1$,
  build 0 true, build 0 false)

  in build (succ(0) true)

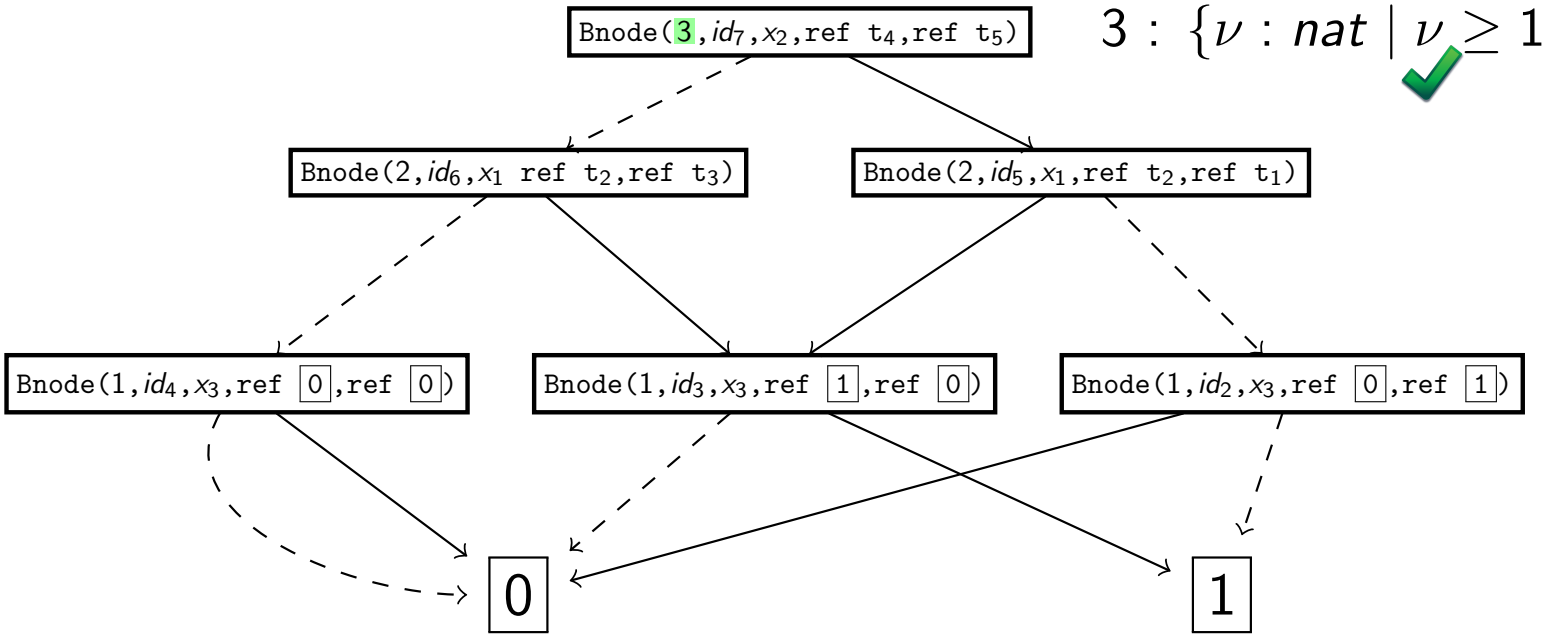


# Type checking and type inference for a 3-level BDD



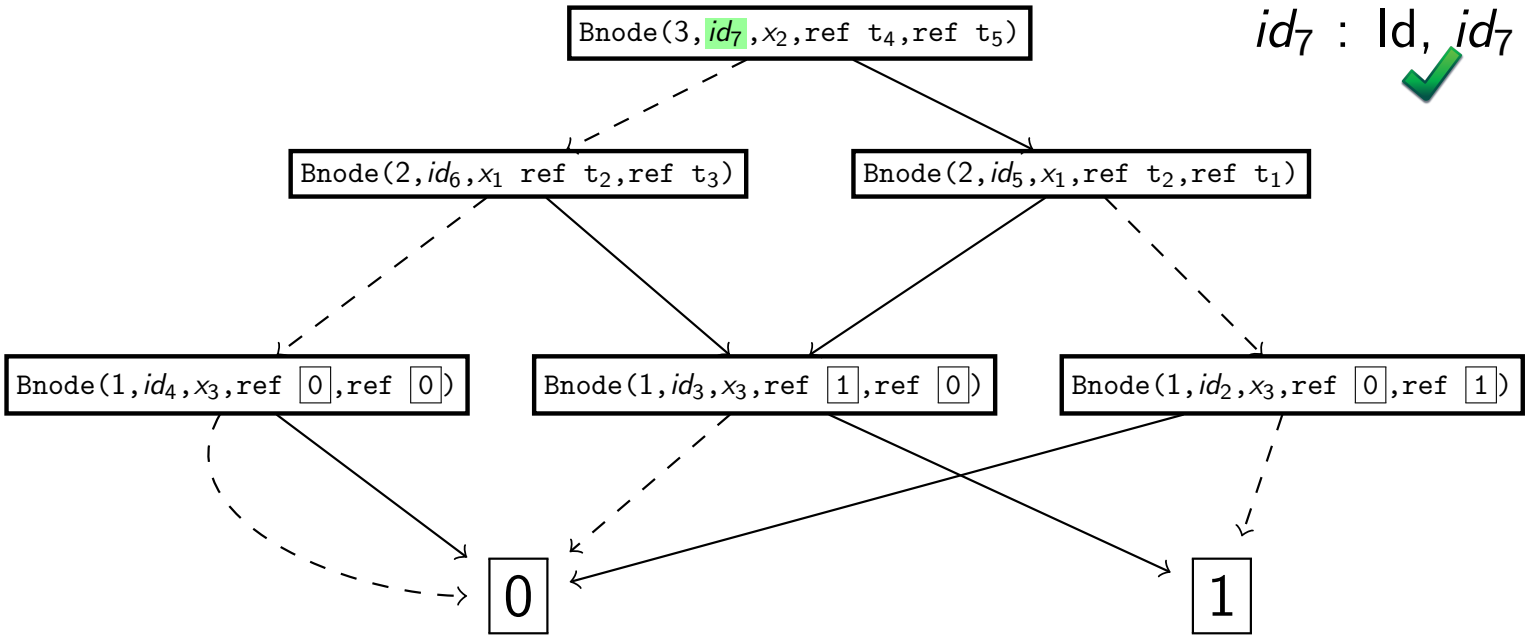

$$\text{T-BNODE}\ \frac{\begin{array}{c}\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : string \quad \Gamma \vdash \mathtt{t}_0 : \{\nu : nat \mid \nu \geq 1 \wedge \nu = l\} \\ \Gamma \vdash \mathtt{t}_1 : \mathtt{ref}\ \ bdd[l', r, c] \quad bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\ \Gamma \vdash \mathtt{t}_2 : \mathtt{ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}\end{array}}{\Gamma, id : Id \vdash \mathtt{Bnode}(\mathtt{t}_0, id, v_{var}, \mathtt{t}_1, \mathtt{t}_2) : bdd[l, r, c]}$$

# Type checking and type inference for a 3-level BDD




$$\text{T-BNODE}\ \frac{\begin{array}{c}\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : string \quad \Gamma \vdash t_0 : \{\nu : nat \mid \nu \geq 1 \wedge \nu = l\} \\ \Gamma \vdash t_1 : \texttt{ref}\ \ bdd[l', r, c] \quad bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\ \Gamma \vdash t_2 : \texttt{ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}\end{array}}{\Gamma, id : Id \vdash \texttt{Bnode}\ (t_0, id, v_{var}, t_1, t_2) : bdd[l, r, c]}$$

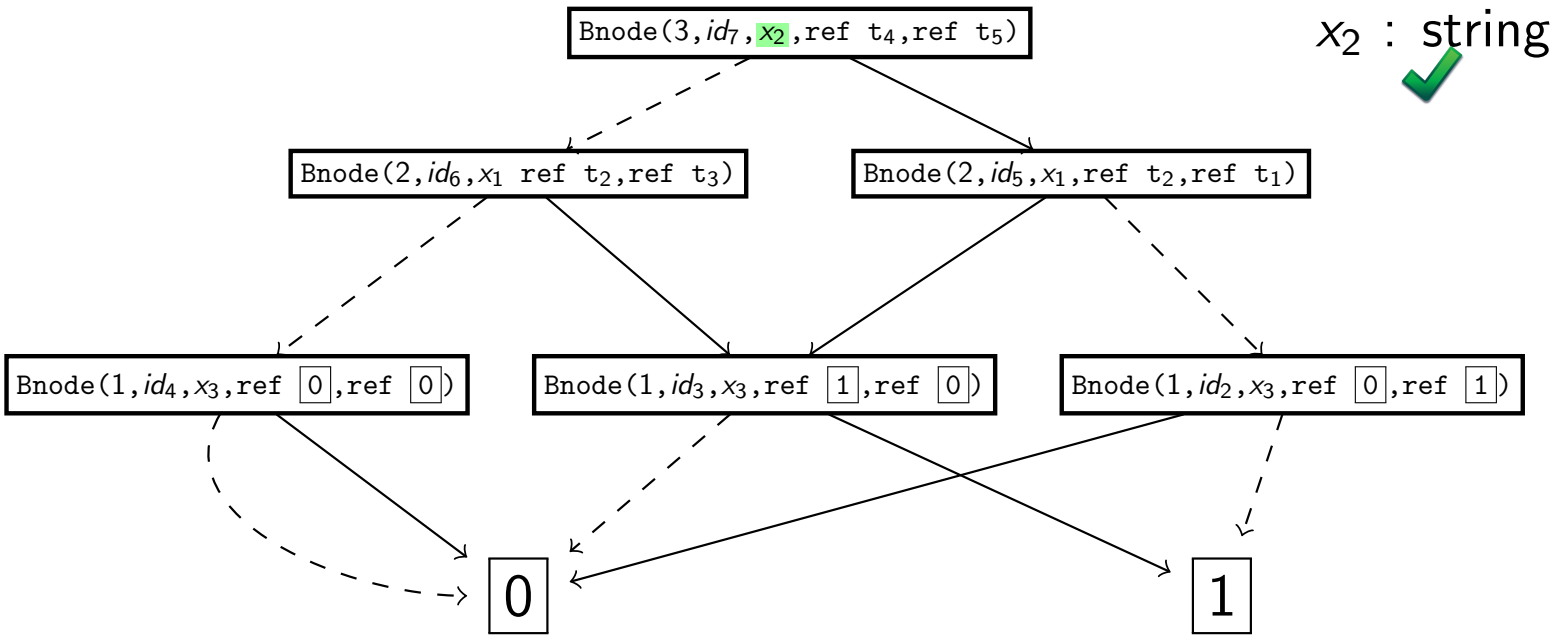
# Type checking and type inference for a 3-level BDD




$id_7 : \mathsf{Id},\ id_7 \notin \Gamma$

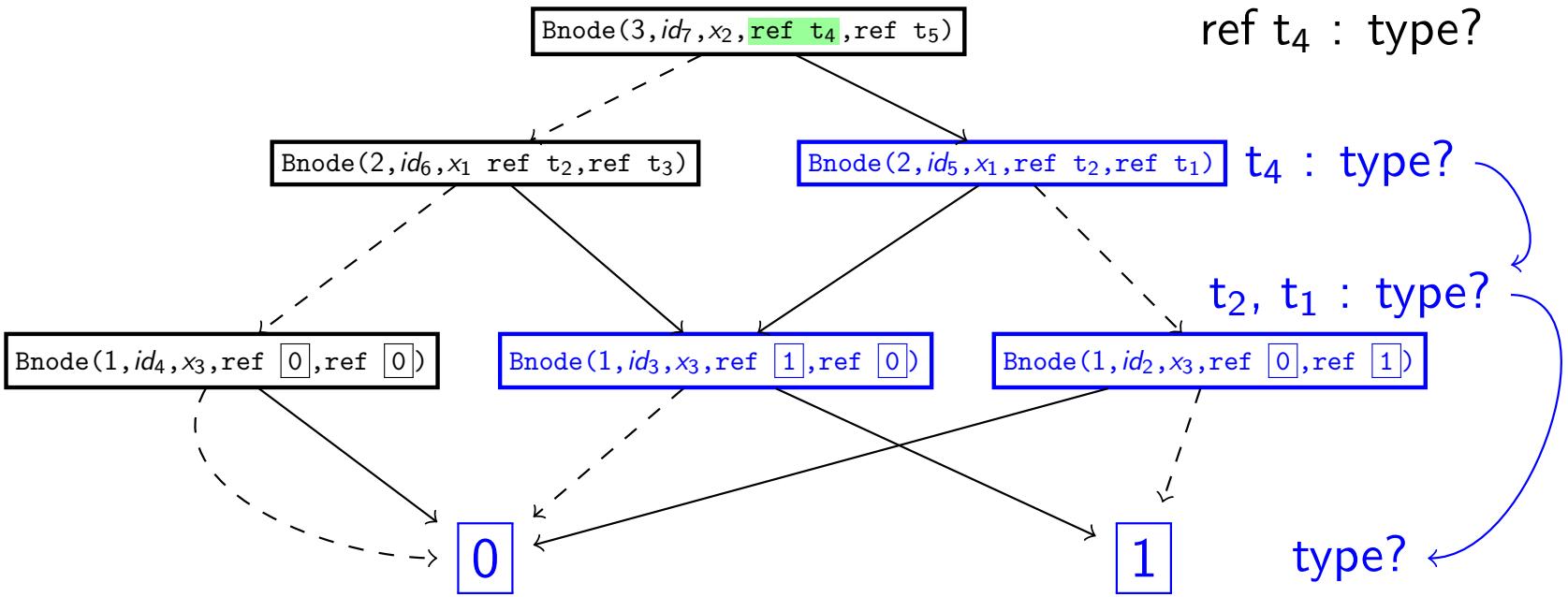
$$\textsc{T-Bnode}\ \frac{\begin{array}{c}\Gamma \vdash \mathsf{id}:\mathsf{Id} \quad \mathsf{id} \notin \mathsf{dom}(\Gamma) \quad \Gamma \vdash v_{var} : string \quad \Gamma \vdash \mathtt{t}_0 : \{\nu : nat \mid \nu \geq 1 \wedge \nu = l\} \\ \Gamma \vdash \mathtt{t}_1 : \mathtt{ref}\ \ bdd[l', r, c] \quad bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\ \Gamma \vdash \mathtt{t}_2 : \mathtt{ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}\end{array}}{\Gamma, id : Id \vdash \mathtt{Bnode}\ (\mathtt{t}_0, id, v_{var}, \mathtt{t}_1, \mathtt{t}_2) : bdd[l, r, c]}$$

# Type checking and type inference for a 3-level BDD



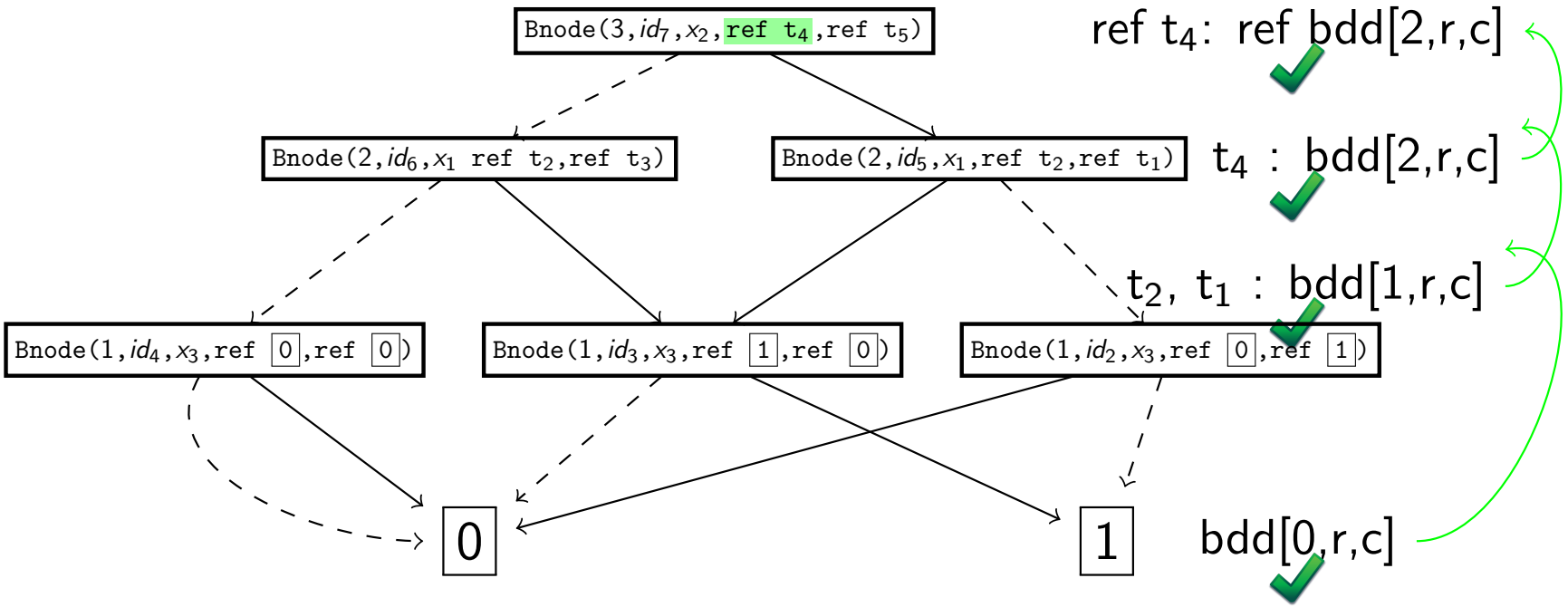

$$\textsc{T-Bnode}\ \frac{\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : \text{string} \quad \Gamma \vdash \mathtt{t}_0 : \{\nu : nat \mid \nu \geq 1 \wedge \nu = l\} \quad \Gamma \vdash \mathtt{t}_1 : \mathtt{ref}\ \ bdd[l', r, c] \quad bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}}{\Gamma, id : Id \vdash \mathtt{Bnode}\ (\mathtt{t}_0, id, v_{var}, \mathtt{t}_1, \mathtt{t}_2) : bdd[l, r, c]}$$

# Type checking and type inference for a 3-level BDD



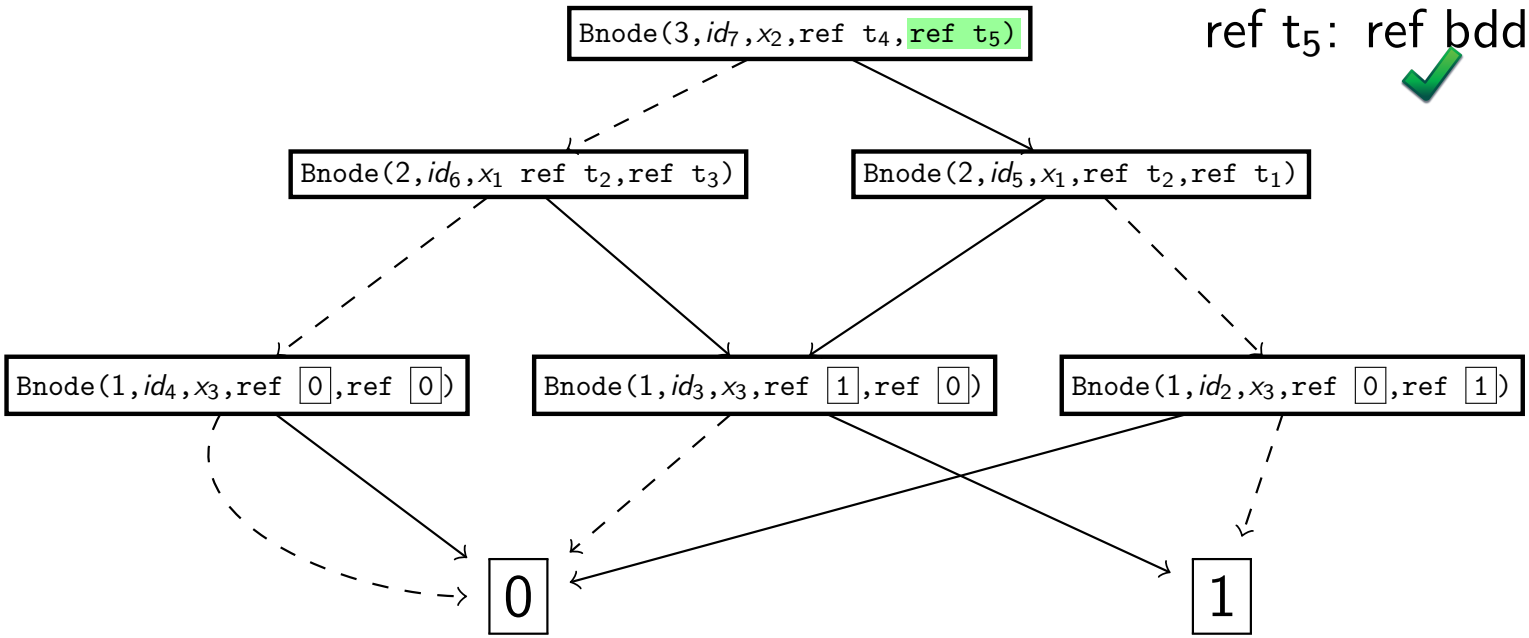

$$\textsc{T-Bnode}\ \frac{\begin{array}{c}\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : string \quad \Gamma \vdash \mathtt{t}_0 : \{\nu : nat \mid \nu \geq 1 \wedge \nu = l\} \\ \Gamma \vdash \mathtt{t}_1 : \mathtt{ref}\ \ bdd[l', r, c] \quad bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\ \Gamma \vdash \mathtt{t}_2 : \mathtt{ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}\end{array}}{\Gamma, id : Id \vdash \mathtt{Bnode}\ \ (\mathtt{t}_0, id, v_{var}, \mathtt{t}_1, \mathtt{t}_2) : bdd[l, r, c]}$$

# Type checking and type inference for a 3-level BDD




$$\textsc{T-Bnode}\ \frac{\begin{array}{c}\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : string \quad \Gamma \vdash \mathtt{t}_0 : \{\nu : nat \mid \nu \geq 1 \wedge \nu = l\} \\ \Gamma \vdash \mathtt{t}_1 : \mathtt{ref}\ \ bdd[l', r, c] \quad bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\ \Gamma \vdash \mathtt{t}_2 : \mathtt{ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}\end{array}}{\Gamma, id : Id \vdash \mathtt{Bnode}\ (\mathtt{t}_0, id, v_{var}, \mathtt{t}_1, \mathtt{t}_2) : bdd[l, r, c]}$$

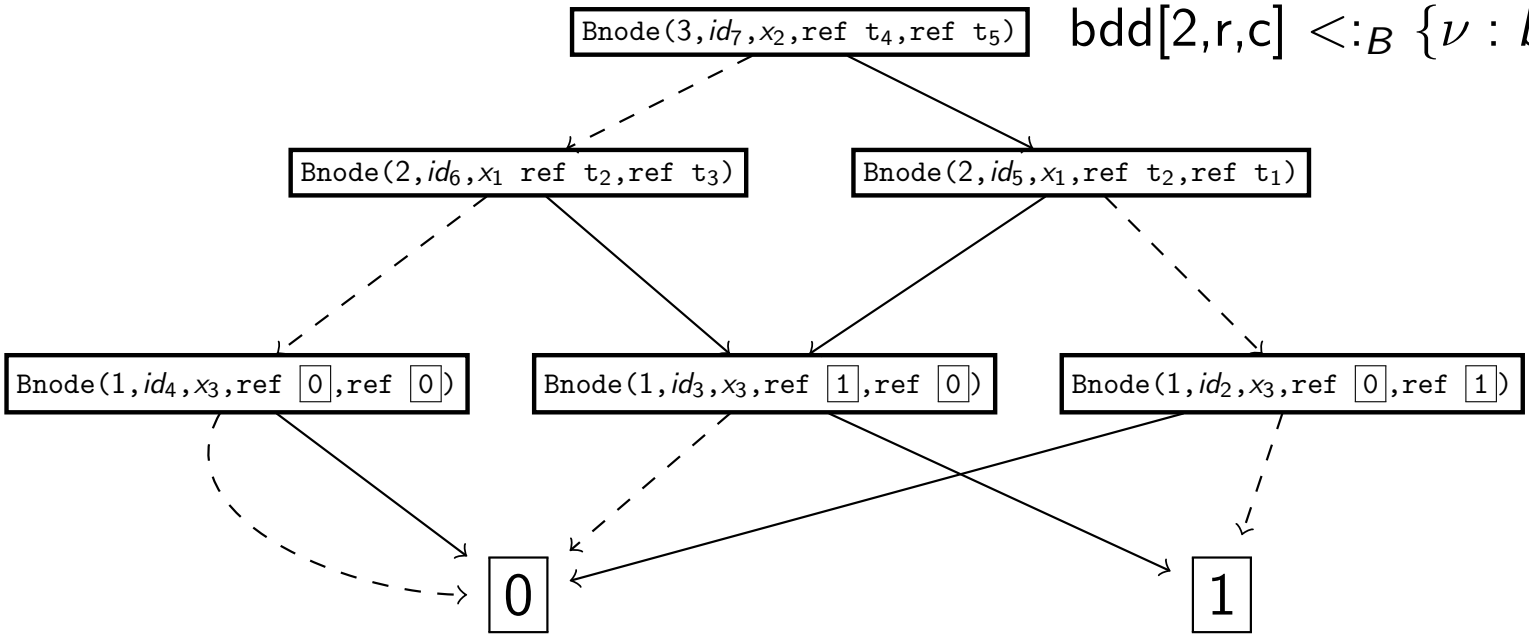
# Type checking and type inference for a 3-level BDD




ref $t_5$: ref bdd[2,r,c]

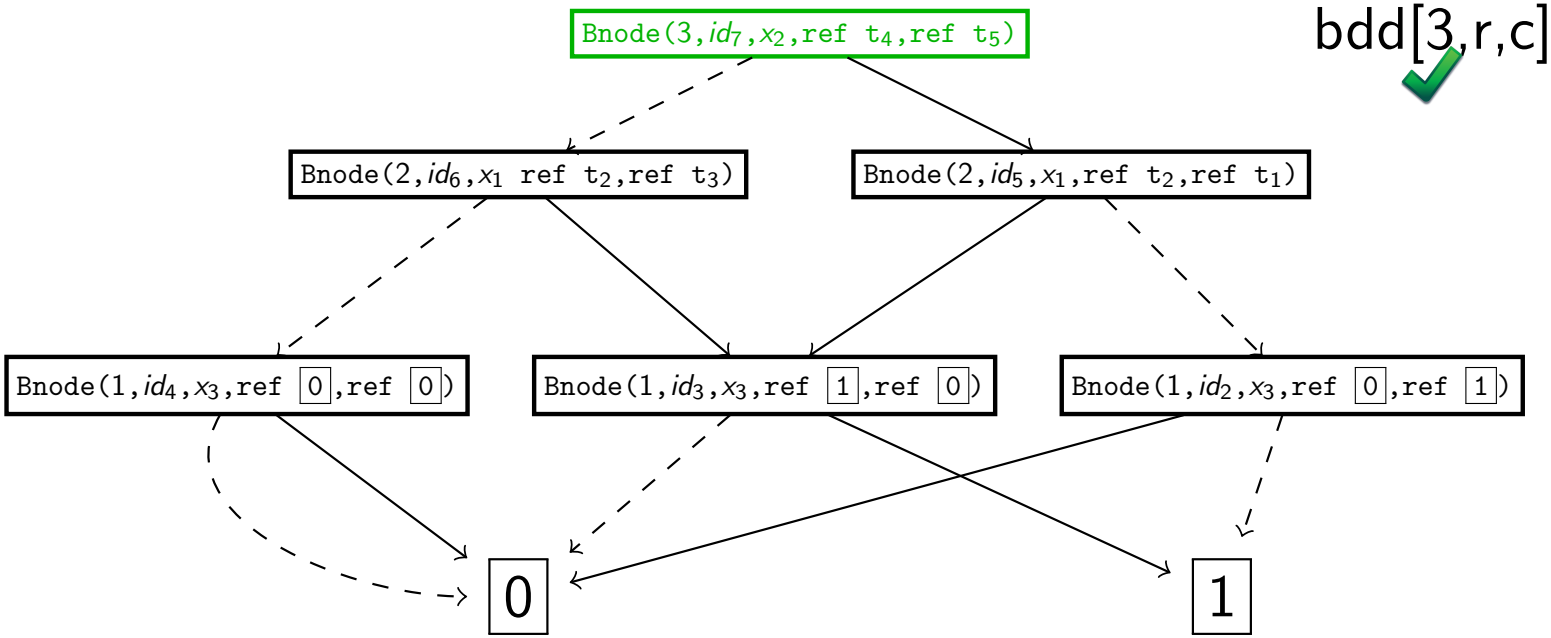
$$\text{T-BNODE}\ \frac{\begin{array}{c}\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : string \\ \Gamma \vdash t_1 : \texttt{ref}\ \ bdd[l', r, c] \quad bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\ \Gamma \vdash t_2 : \texttt{ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}\end{array}}{\Gamma, id : Id \vdash \texttt{Bnode}\ \ (t_0, id, v_{var}, t_1, t_2) : bdd[l, r, c]}$$

# Type checking and type inference for a 3-level BDD




$$\text{T-BNODE}\ \frac{\begin{array}{c}\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : string \\ \Gamma \vdash \mathtt{t_1{:}ref}\ \ bdd[l', r, c] \quad \text{bdd[l',r,c]} <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\ \Gamma \vdash \mathtt{t_2{:}ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}\end{array}}{\Gamma, id : Id \vdash \mathtt{Bnode}\ (\mathtt{t_0}, id, v_{var}, \mathtt{t_1}, \mathtt{t_2}) : bdd[l, r, c]}$$

# Type checking and type inference for a 3-level BDD




$$\text{T-BNODE}\ \frac{\begin{array}{c}\Gamma \vdash id : Id \quad id \notin dom(\Gamma) \quad \Gamma \vdash v_{var} : string \\ \Gamma \vdash \mathtt{t_1{:}ref}\ \ bdd[l', r, c] \quad bdd[l', r, c] <:_B \{\nu : bdd[p, r, c] \mid l = p + 1\} \\ \Gamma \vdash \mathtt{t_2{:}ref}\ \ bdd[l'', r, c] \quad bdd[l'', r, c] <:_B \{\nu : bdd[p', r, c] \mid l = p' + 1\}\end{array}}{\Gamma, id : Id \vdash \mathtt{Bnode}\ (\mathtt{t_0}, id, v_{var}, \mathtt{t_1}, \mathtt{t_2}) : \text{bdd[l,r,c]}}$$

# BDDL in Action: Runtime Error Prevention in CUDD

```
DdNode* Cudd_bddCompose(
... // bdd composition
}

int main () {
  Cudd mgr(0,2);
  BDD x = mgr.bddVar();
  BDD y = mgr.bddVar();
  BDD h = x * y;
  BDD j = x + y;
  BDD k = h.Compose(j,2);
}
```

✗

$\Longrightarrow$

```
letrec two : {ν : nat | ν = 2} =
(succ (succ 0)) in
letrec bddCompose : bdd[l, r, c]
→ bdd[l', r, c]→ {ν : nat | ν ≤ l − 1}
  → bdd[l'', r, c] =
   λ f .
     λ g .
       λ v .
       <body>
in
letrec h = Bnode(two, ...)  in
letrec j = ...  in
letrec k =
  bddCompose h j two ;
```

# BDDL in Action: Runtime Error Prevention in CUDD

```
DdNode* Cudd_bddCompose(
... // bdd composition
}

int main () {
  Cudd mgr(0,2);
  BDD x = mgr.bddVar();
  BDD y = mgr.bddVar();
  BDD h = x * y;
  BDD j = x + y;
  BDD k = h.Compose(j,2);
}
```

```
letrec two : {ν : nat | ν = 2} = (succ (succ 0)) in
letrec bddCompose : bdd[l, r, c] → bdd[l', r, c] → {ν : nat | ν ≤ l − 1} →
  bdd[l'', r, c] =
  λ f .
    λ g .
      λ v .
      <body>
in
letrec h : bdd[2,r,c] = ... in
letrec j : bdd[2,r,c] = ... in
letrec k =
   bddCompose h j two
```

Runtime time error

$\{\nu : nat \mid \nu = 2\} \neq \{\nu : nat \mid \nu \leq 1\}$

Compile time error

# BDDL in Action: Runtime Error Prevention in MDDL

```
BddNode* Union_QQ(
 BddNode *p, BddNode *q){
 ASSERT((k = p->GetLevel()))
 == q->GetLevel());
  ...
  ka = answer→GetLevel();
  ASSERT(ka == k);
  return answer;
}
int main ()
{
  BddNode *f = new_bdd(1);
  BddNode *g = new_bdd(2);
  BddNode *res = Union_QQ(f,g);

}
```

```
letrec union :
  bbd[l, q, c]->
    bbd[l, q, c] ->
      bbd[l, q, c]) =
λ p .  λ r .  <body>
in letrec f :  bbd[1, q, s] = ...
in letrec g :  bbd[2, q, s] = ...
in union f g
```

# BDDL in Action: Runtime Error Prevention in MDDL

```
BddNode* Union_QQ(
 BddNode *p, BddNode *q){
 ASSERT((k = p->GetLevel()))
 == q->GetLevel());
  ...
  ka = answer→GetLevel();
  ASSERT(ka == k);
  return answer;
}
int main ()
{
  BddNode *f = new_bdd(1);
  BddNode *g = new_bdd(2);
  BddNode *res = Union_QQ(f,g);

}
```

Runtime time error

$\Longrightarrow$

```
letrec union :
  bdd[l, q, c] ->✗
    bdd[l, q, c] ->✗
      bdd[l, q, c]) =
λ p .  λ r .  <body>
in letrec f : bdd[1, q, s]✗ = ...
in letrec g : bdd[2, q, s]✗ = ...
in union f g
```

$bdd[1, q, s] \neq bdd[2, q, s]$

Compile time error

# BDDL in Action: Runtime Error Prevention in JavaBDD

```
public class BasicTests
extends BDDTestCase {...
  public void testCrash() {
    reset();
    Assert.assertTrue(hasNext());
    BDDFactory bdd = nextFactory();
    BDD a = bdd.one();
    bdd.reorder(
          bdd.getReorderMethod());
}
```

```
letrec n : bdd[0,r,c] = 1
in letrec reorder :
{ν : bbd[l, r, c] | l ≥ 1}
→ {ν : bbd[l, r, c] | l ≥ 1} =
  λ p : bbd[l, r, c]. <body>
in
  reorder n
```

# BDDL in Action: Runtime Error Prevention in JavaBDD

```
public class BasicTests
extends BDDTestCase {...
  public void testCrash() {
    reset();
    Assert.assertTrue(hasNext());
    BDDFactory bdd = nextFactory();
    BDD a = bdd.one();
    bdd.reorder(
          bdd.getReorderMethod());
}
```

✗

Runtime time error

$\Longrightarrow$

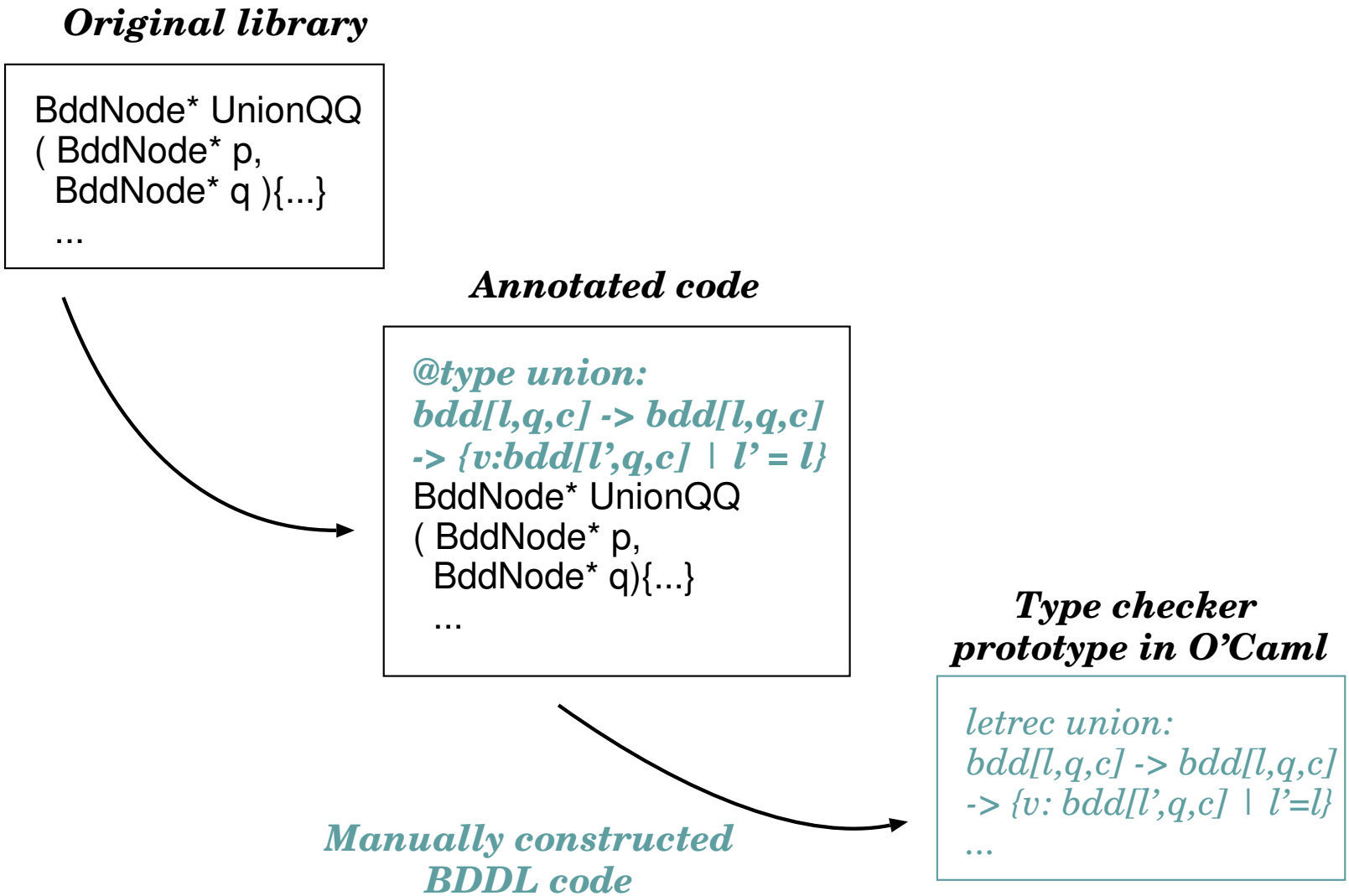
```
letrec n :  bdd[0,r,c] = 1
in letrec reorder :
{ν : bdd[l, r, c] | l ≥ 1}
→ {ν : bdd[l, r, c] | l ≥ 1} =
  λ p :  bdd[l, r, c].  <body>
in
  reorder n
```

✗

$bdd[0, r, c] \neq \{bdd[l, r, c] | l \geq 1\}$

Compile time error

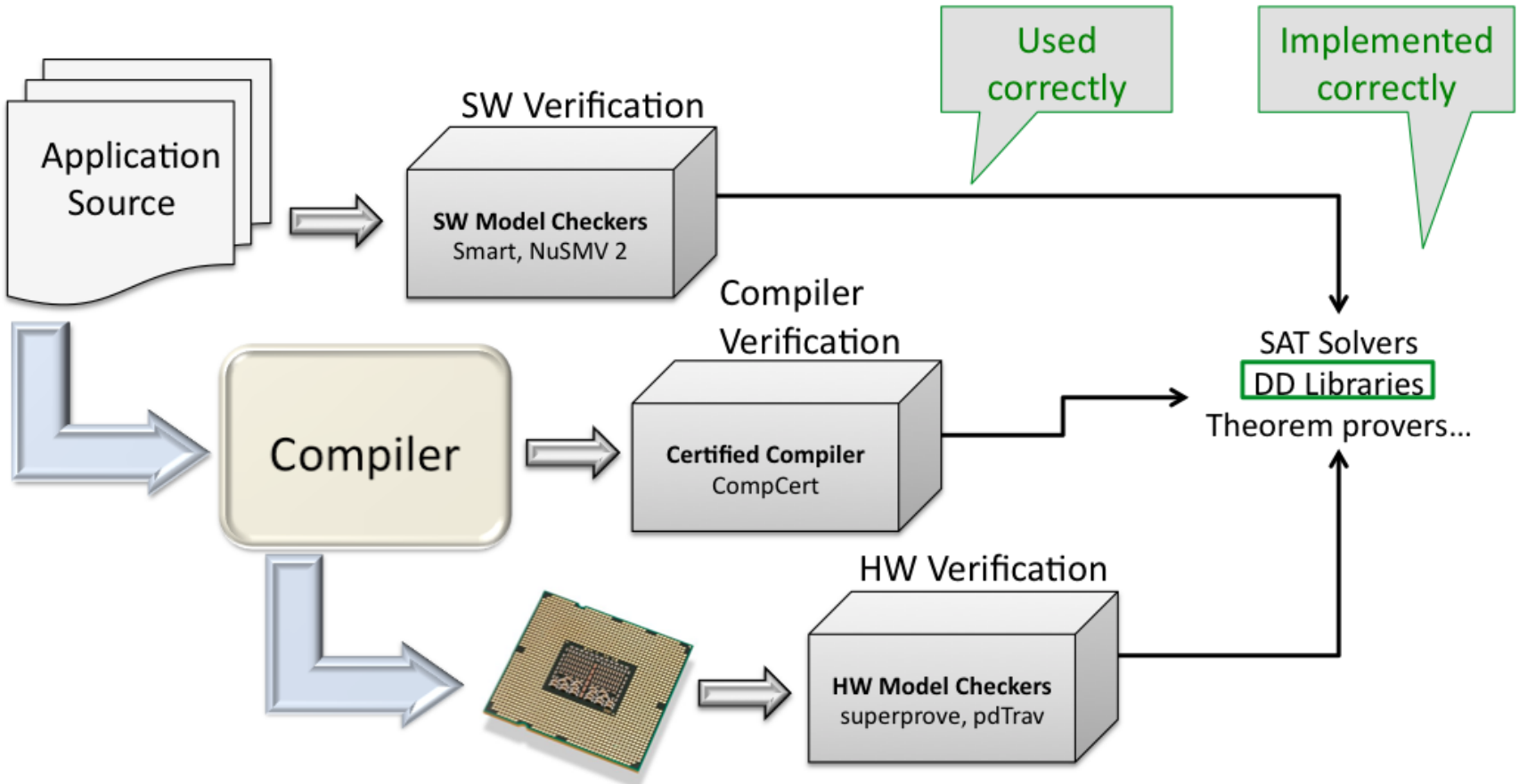
# Our approach and future directions



# Some Related Work

- Static analysis
  -  Lhoták, Ondřej, and Laurie Hendren
    Jedd: a BDD-based relational extension of Java.
    PLDI, 2004.
- Runtime verification
  -  Rolf Drechsler
    Verifying integrity of decision diagrams.
    Integration, the VLSI Journal, 2002.
- Liquid types
  -  Ming Kawaguchi, Patrick Rondon, and Ranjit Jhala
    Type-based data structure verification.
    PLDI, 2009.

# Conclusion


Application Source
SW Verification
SW Model Checkers
Smart, NuSMV 2
Used correctly
Implemented correctly
Compiler
Compiler Verification
Certified Compiler
CompCert
SAT Solvers
DD Libraries
Theorem provers...
HW Verification
HW Model Checkers
superprove, pdTrav