A decision procedure for alternation-free modal -calculi

Yoshinori Tanabe[1] Koichi Takahashi[2] Masami Hagiya[1]

1: University of Tokyo

2: National Institute of Advanced Industrial Science and Technology

Advanced in Modal Logic 2008

Motivation

Analysis of variants of graph rewriting systems
using decision procedures for satis ability of temporal/m odal logics

Previous work:

- Decision Procedure for two-way CTL [Hagiya et al 04]
  - Applied to analysis of one-dimensional cellular automata.
- Decision Procedure for alternation-free two-way -calculus [Tanabe et al 05]
  - Applied to analysis of XML transducers.

This research:

- Decision Procedure: (sublogics of) alternation-free + nominals + backward modalities + functional modalities.
  (variants of enriched -calculus)
- Target: analysis of programs that manipulates pointers.

Motivation

Analysis of variants of graph rewriting systems
using decision procedures for satis ability of temporal/m odal logics
Previous work:

- Decision Procedure for two-way CTL [Hagiya et al 04]
  - Applied to analysis of one-dimensional cellular automata.
- Decision Procedure for alternation-free two-way -calculus [Tanabe et al 05]
  - Applied to analysis of XML transducers.

This research:

- Decision Procedure: (sublogics of) alternation-free + nominals + backward modalities + functional modalities.
  (variants of enriched -calculus)
- Target: analysis of programs that manipulates pointers.

Motivation

Analysis of variants of graph rewriting systems
using decision procedures for satis ability of temporal/m odal logics
Previous work:

- Decision Procedure for two-way CTL [Hagiya et al 04]
  - Applied to analysis of one-dimensional cellular automata.
- Decision Procedure for alternation-free two-way -calculus [Tanabe et al 05]
  - Applied to analysis of XML transducers.

This research:

- Decision Procedure: (sublogics of) alternation-free + nominals + backward modalities + functional modalities.
  (variants of enriched -calculus)
- Target: analysis of programs that manipulates pointers.

Motivation

Analysis of variants of graph rewriting systems
using decision procedures for satis ability of temporal/m odal logics
Previous work:

- Decision Procedure for two-way CTL [Hagiya et al 04]
  - Applied to analysis of one-dimensional cellular automata.
- Decision Procedure for alternation-free two-way -calculus [Tanabe et al 05]
  - Applied to analysis of XML transducers.

This research:

- Decision Procedure: (sublogics of) alternation-free + nominals + backward modalities + functional modalities.
  (variants of enriched -calculus)
- Target: analysis of programs that manipulates pointers.

Known results on satis ability

L ($_{AF}$)	EXPTIME-complete	[Emerson 90]
L ($_{AF}$; nom; back, func)	Undecidable	[Bonatti, Peron 04]
L (; nom; back)	EXPTIME-complete	[Sattler, Vardi 01]
L (; nom; func)	EXPTIME-complete	[Bonatti et al 06]
L (; back, func)	EXPTIME-complete	[Bonatti et al 06]
L (nom; back, func)	NEXPTIME-complete	[Tobies 00]

- : (full) -calculus
- $_{AF}$: alternation-free fragment of -calculus
- nom: nominals
- back: backward modalities
- func: functional modalities

Our decision procedure

Logic	Time Complexiy
L ($_{AF}$; nom; back)	$2^{O(n^2 \log n)}$
L ($_{AF}$; nom; func)	$2^{O(n^2 \log n)}$
L ($_{AF}$; back; func)	$2^{O(n \log n)}$

- Existing procedures are automata-based.
  - Powerful: no restriction for alternation
  - Complex: $2^{O(n^4)}$ even for L ( ).
- Ours is by simply enumerating possible nodes.
  - Only for alternation-free fragments.
  - Allows ef cient implementation using BDDs.
  - Has successfully been applied to the target.

Syntax

PS 3 p	Propositional Symbols
Nom 3 x	Nominals
PV 3 X	Propositional Variables
FMS 3 f	Funcitonal Modality Symbols
GMS 3 g	General Modality Symbols
MS 3 h ::= f j g	Modality Symbols
Mod 3 m ::= h j h $^{-1}$	Modalities
Form 3 '	Formulas

::= p j x j X j : ' j ' _ ' j hmi ' j X ' j @x '

Semantics

Kripke Structure $K = (S; R; L)$

- S: Set of states
- $R : MS ! P (S S)$
  -

Alternation-freeness

- Alternation-freeness is roughly equal to " - nest-freeness".
- Precisely, ' is alternation-free if it does not have patterns:

  ' = X(Y(X))

  or

  ' = X(Y(X))

- L ($_{AF}$) allows much simpler decision procedure than L ( ).
- It still has suf cient expressive power. Examples:
  - X(p _ hmi X): p is reachable by following m.
  - Y(q_1 ^ ([m^0](q_2 ^ [m^0]Y))): q_1 and q_2 holds alternatively along m^0-paths.

Closure

The closure cl(') of a formula ' is approximately the set of its subformulas, except for xed-point operators.

Example: ' = : p _ X(p _ hmi X)

cl(') = f' ;
: p, X(p _ hmi X),
p, p _ hmi X(p _ hmi X),
hmi X(p _ hmi X) g

Decision procedure for L (AF)

Modi cation of the decision procedure for CTL [Emerson 90].
A tableau method. Each node is an element of P(cl(' I)).

Outline

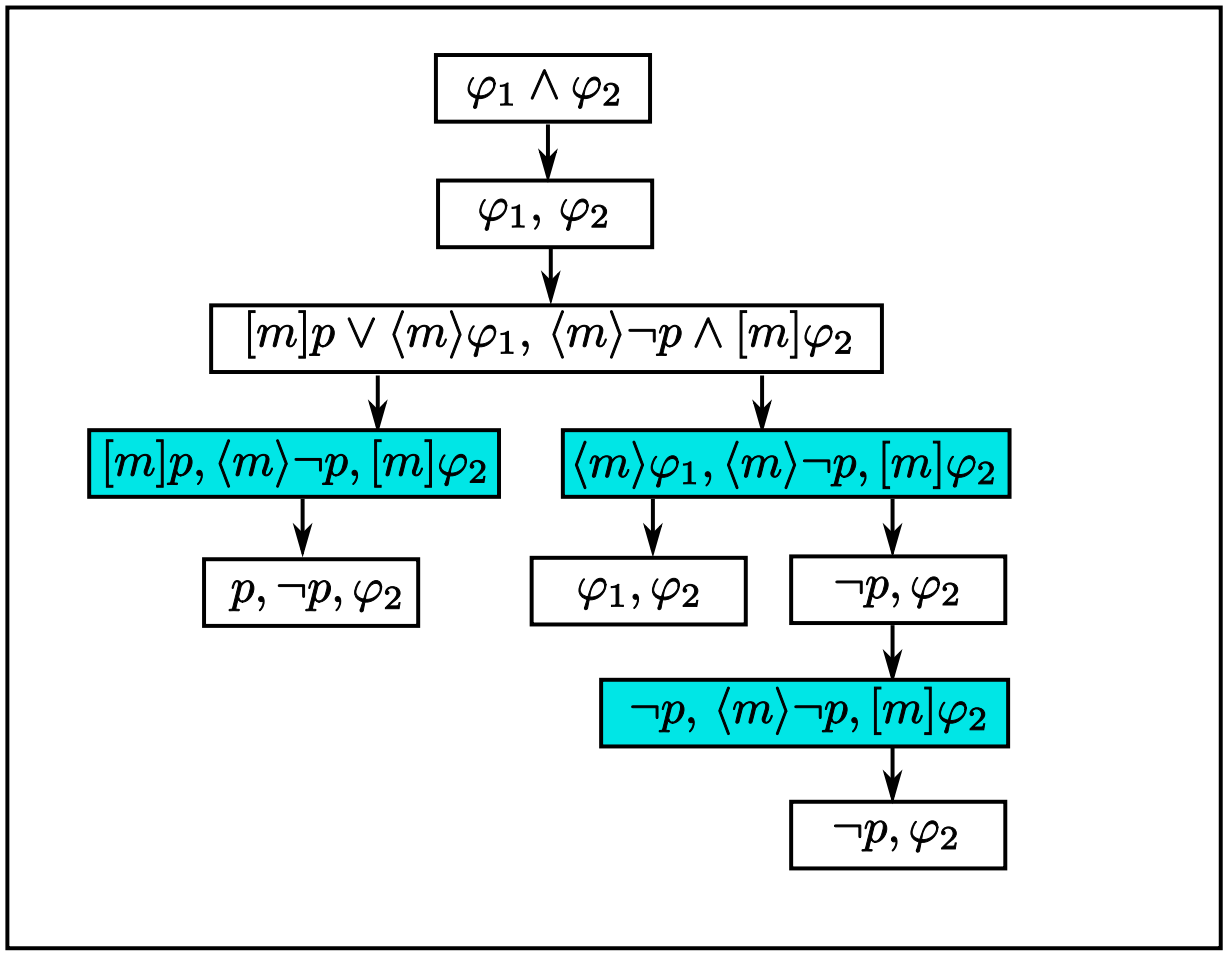
1. Start with initial node n_0 = f' Ig
2. Add nodes by decomposing existing nodes.
3. Remove all inconsistent nodes.
   - : -consistency, -consistency, and -consistency.
4. Repeat the previous step until all remaining nodes are consistent.

' I is satis able iff there remains a node that contains ' I.

Decision procedure for L(AF)

' I = ' 1 ^ ' 2, ' 1 = X([m]p_hmiX), ' 2 = Y(hmi: p^ [m]Y)

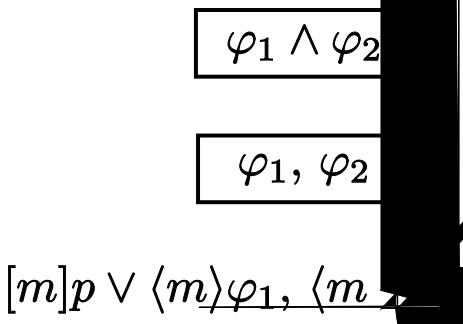
Adding nodes



Decision procedure for L (AP

' I = ' 1 ^ ' 2, ' 1 = X([m]p_hmi ' 2 = Y(hmi: p^[m]Y)

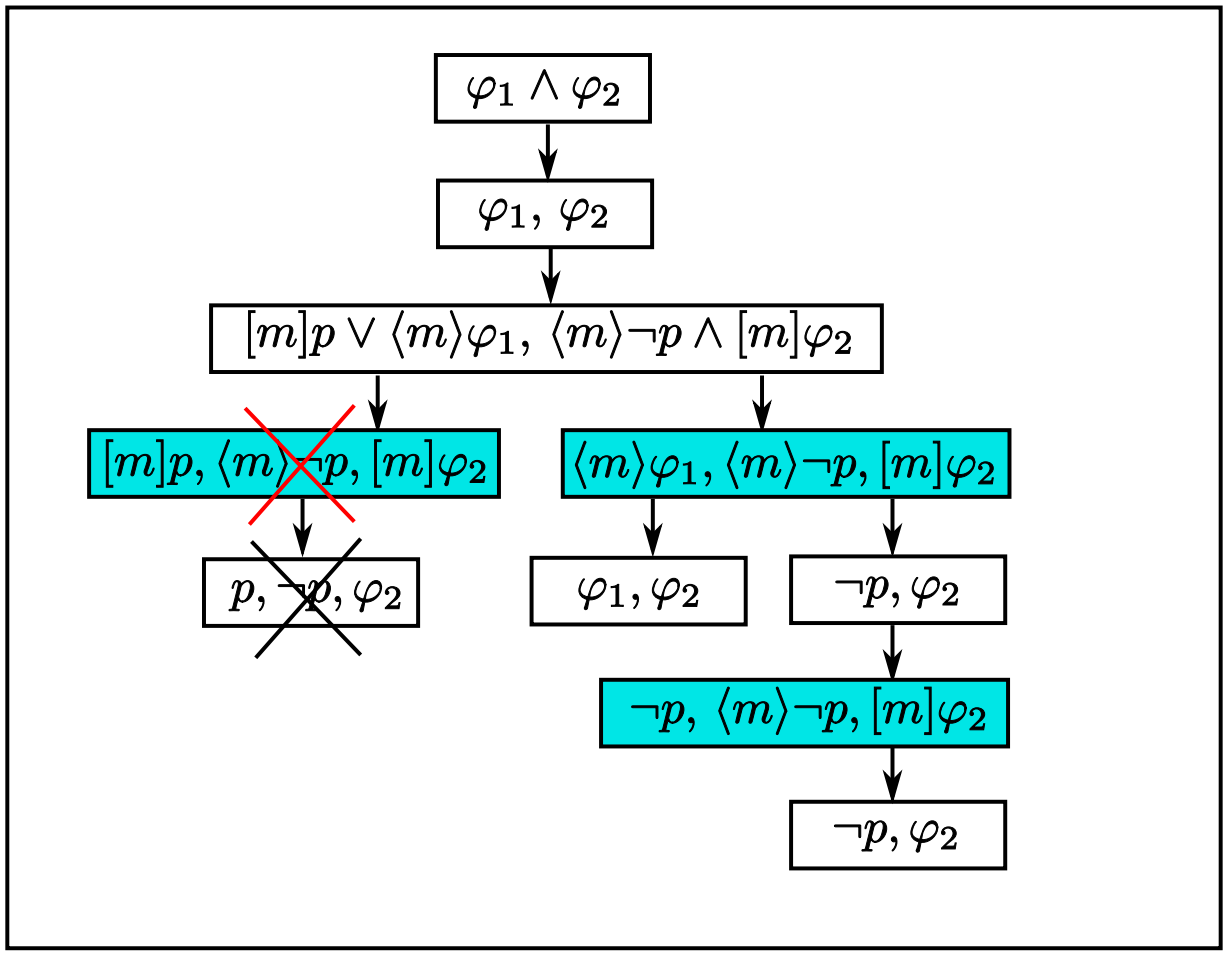
Checking : - nsistency



Decision procedure for L (AF)

' I = ' 1 ^ ' 2, ' 1 = X([m]p _ hmi X), ' 2 = Y(hmi: p ^ [m]Y)

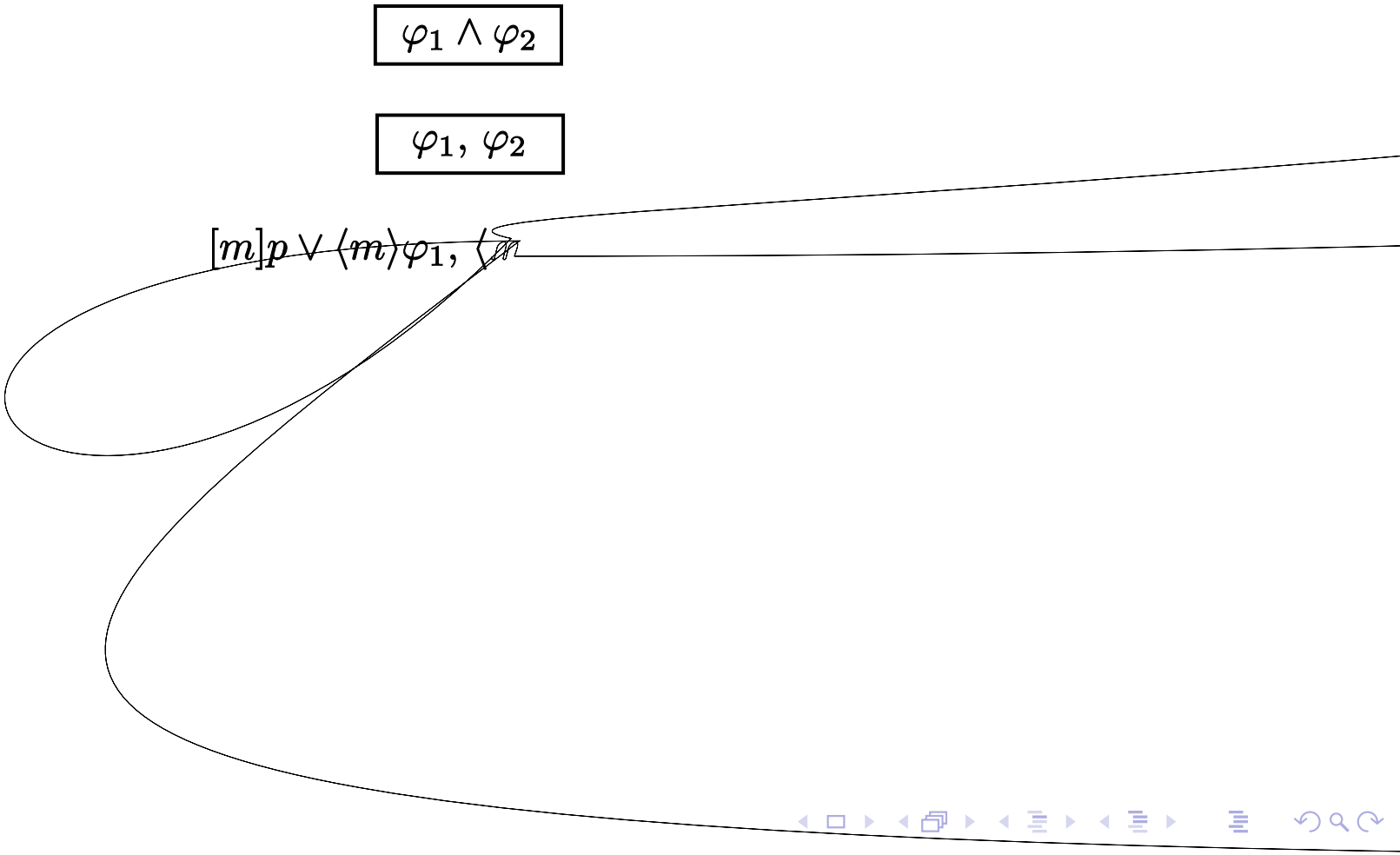
Checking -consistency



Decision procedure for L (AF)

' I = ' 1 ^ ' 2, ' 1 = X([m]p _ hmi X), ' 2 = Y(hmi: p ^ [m]Y)

Checking -consistency – Remove -loops without witnesses

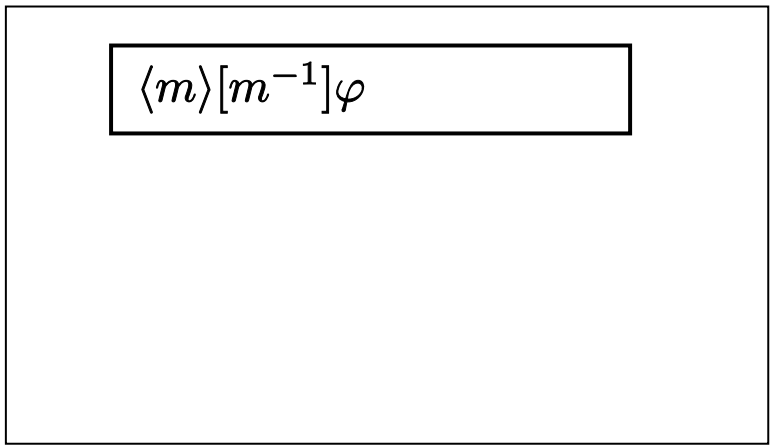


Functional Modality

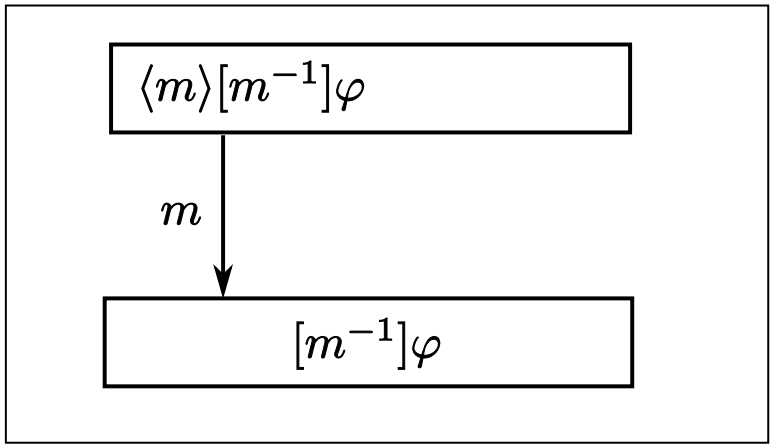
For a node n that contains only atomic formula, -formula and -formula:

- For every formula hgi', where g 2 GMS, add a node and put formulas ' and all where [g] 2 n.
- For every f 2 FMS, if there is hfi' 2 n, add a node and put all formulas where hfi 2 n or [f] 2 n.

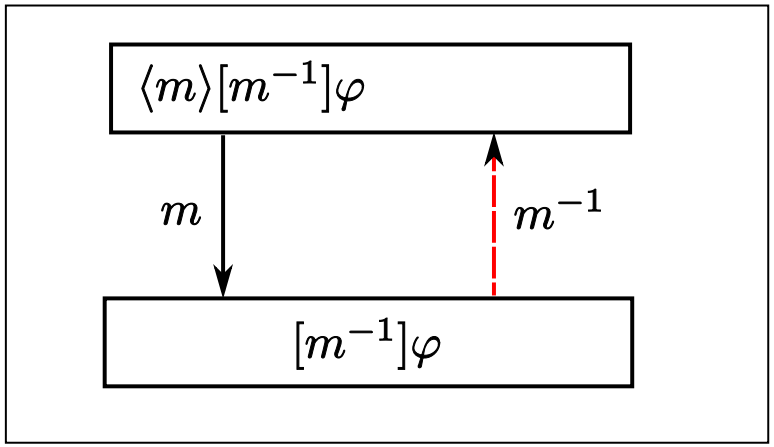
Backward Modality: Problem 1



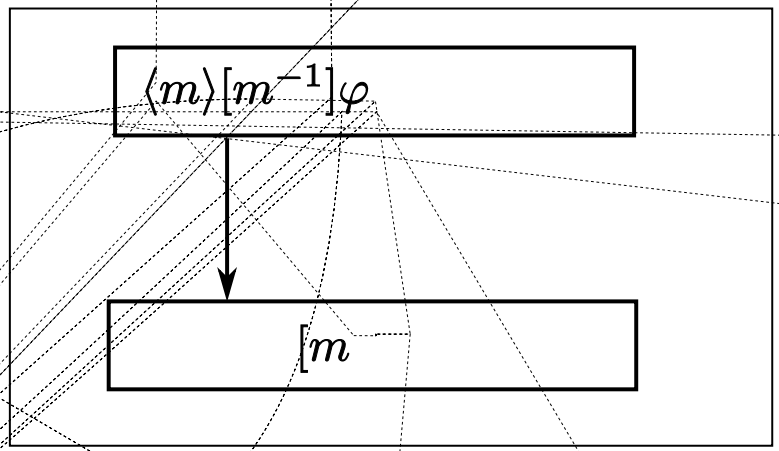
Backward Modality: Problem 1



Backward Modality: Problem 1



Backward Modality: Problem 1



Backward Modality: Solution to Problem 1

Outline of the procedure for given formula ' $_{\mathsf{I}}$:

1. Start with initial node n = f' $_{\mathsf{I}}$g
2. Add nodes by decomposing existing nodes.
3. Remove all inconsistent nodes.
   - : -consistency, -consistency, and -consistency.
4. Repeat the previous step until all remaining nodes are consistent.

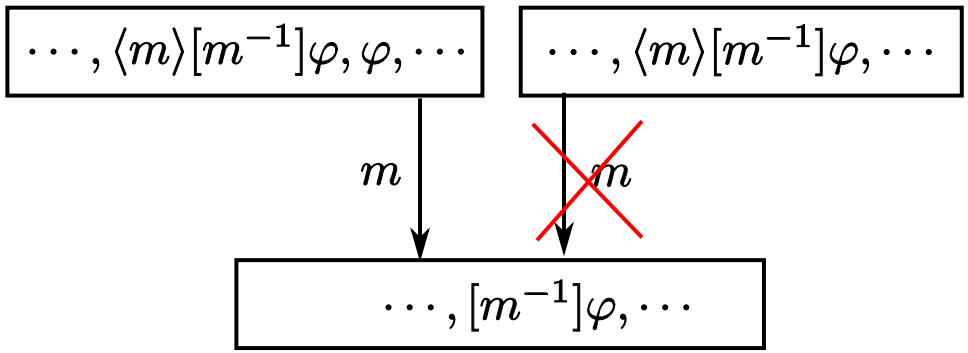
Backward Modality: Solution to Problem 1

Outline of the procedure for given formula ' $_{|}$:

1. Start with all possible nodes.
2. Remove all inconsistent nodes.
   - : -consistency, -consistency, and -consistency.
3. Repeat the previous step until all remaining nodes are consistent.

Backward Modality: Solution to Problem 1

- all possible nodes = all elements of P(cl(' $_{\mathsf{I}}$))
- (n; n^{0}) 2 R(m) ()
  - [m]' 2 n =) ' 2 n^{0}
  - [m ^{1}]' 2 n^{0} =) ' 2 n



Backward Modality: Problem 2

$' _0 = X(\langle m_1 \rangle (\langle m_1^{-1} \rangle p_1 \wedge [m_1^{-1}](\langle m_2 \rangle (\langle m_2^{-1} \rangle p_2 \wedge [m_2^{-1}](p_2 \vee X))))$

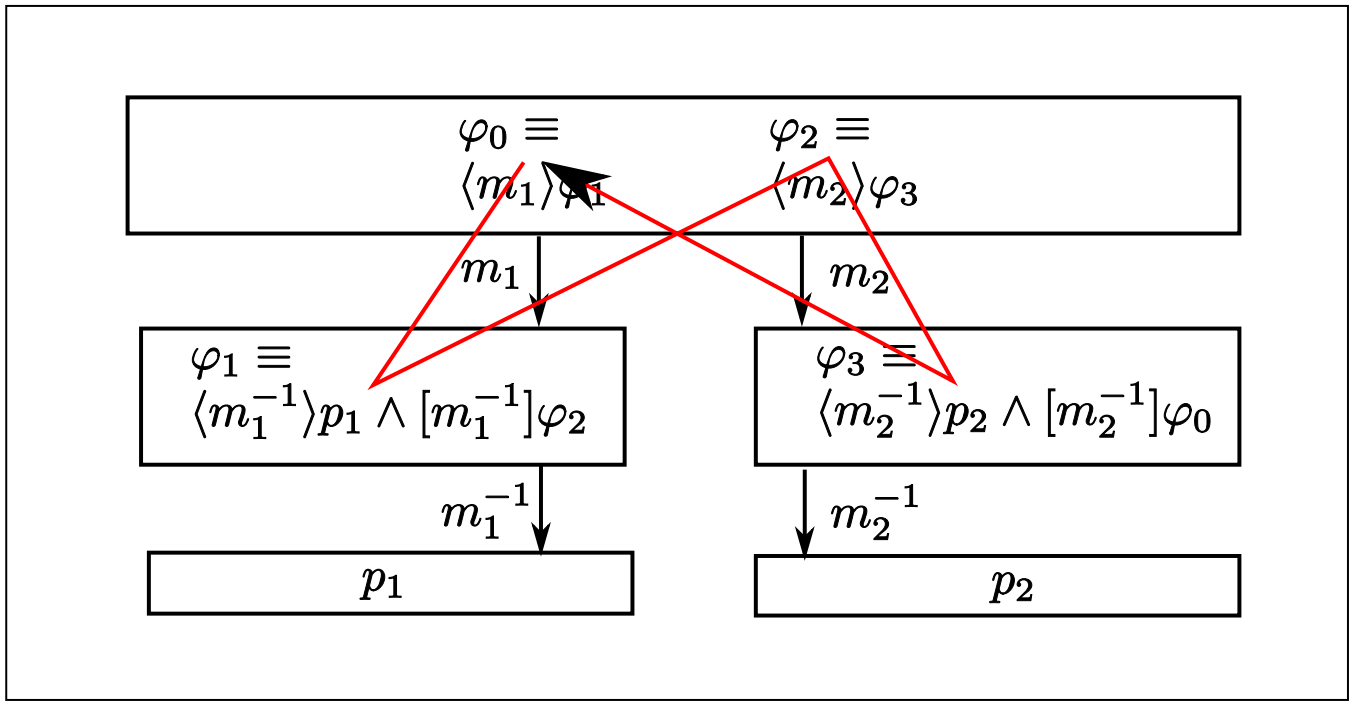
$' _0 \quad \langle m_1 \rangle ' _1, ' _1 = \langle m_1^{-1} \rangle p_1 \wedge [m_1^{-1}] ' _2$
$' _2 = \langle m_2 \rangle ' _3, ' _3 = \langle m_2^{-1} \rangle p_2 \wedge [m_2^{-1}] ' _0$

$' _I = ' _0 \wedge ' _2$: unsatis able formula

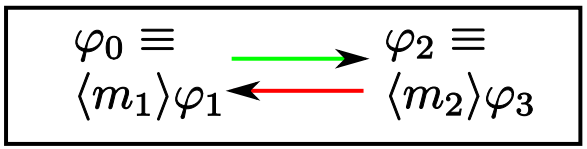
φ_0

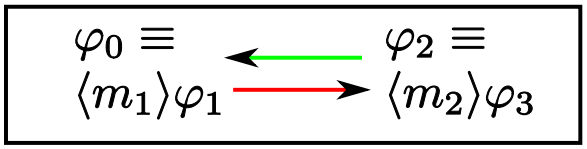
Backward Modality: Problem 2

- All nodes are -consistent.
- However, there is a -loop without witnesses.



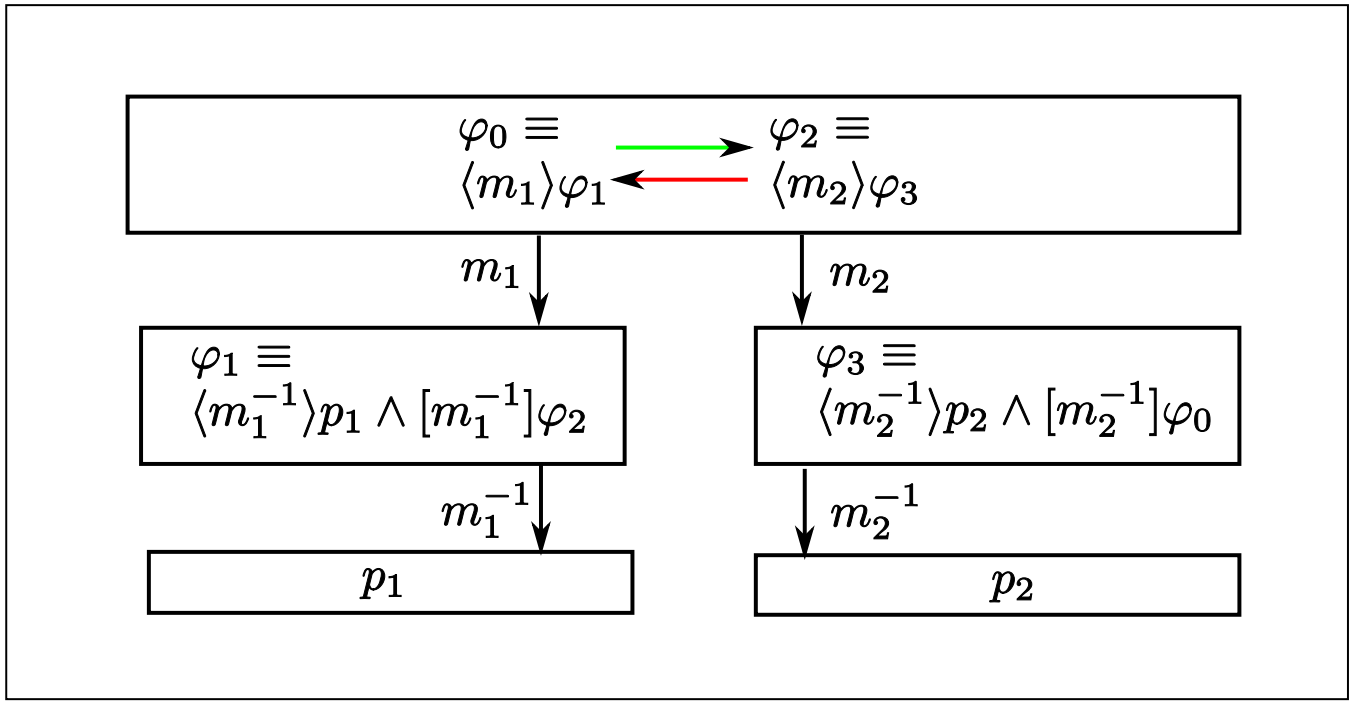
Backward Modality: Solution to Problem 2



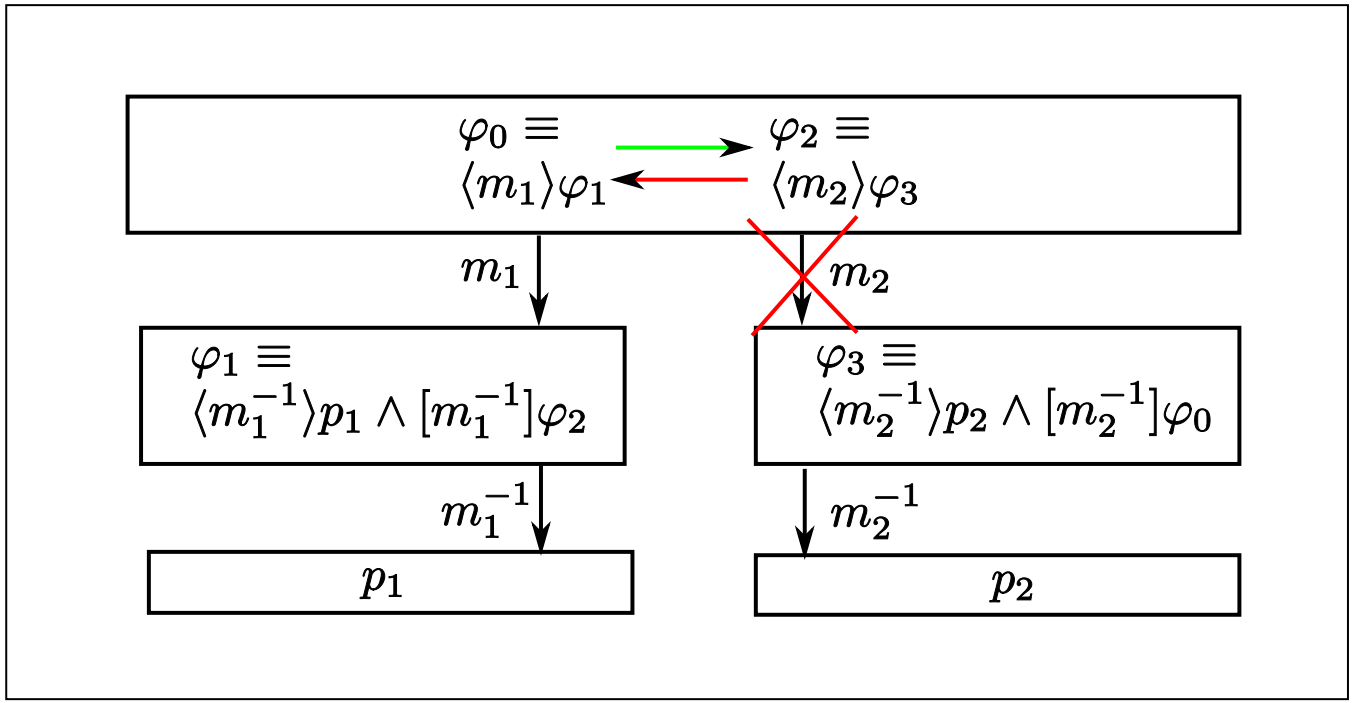


- Each node n is not just a member of P(cl(' $_{\mathrm{I}}$)), but n = (; x), where
  - cl(' $_{\mathrm{I}}$)
  - x carries information on “expansion direction” (indicated by green and red arrows above) for -formulas

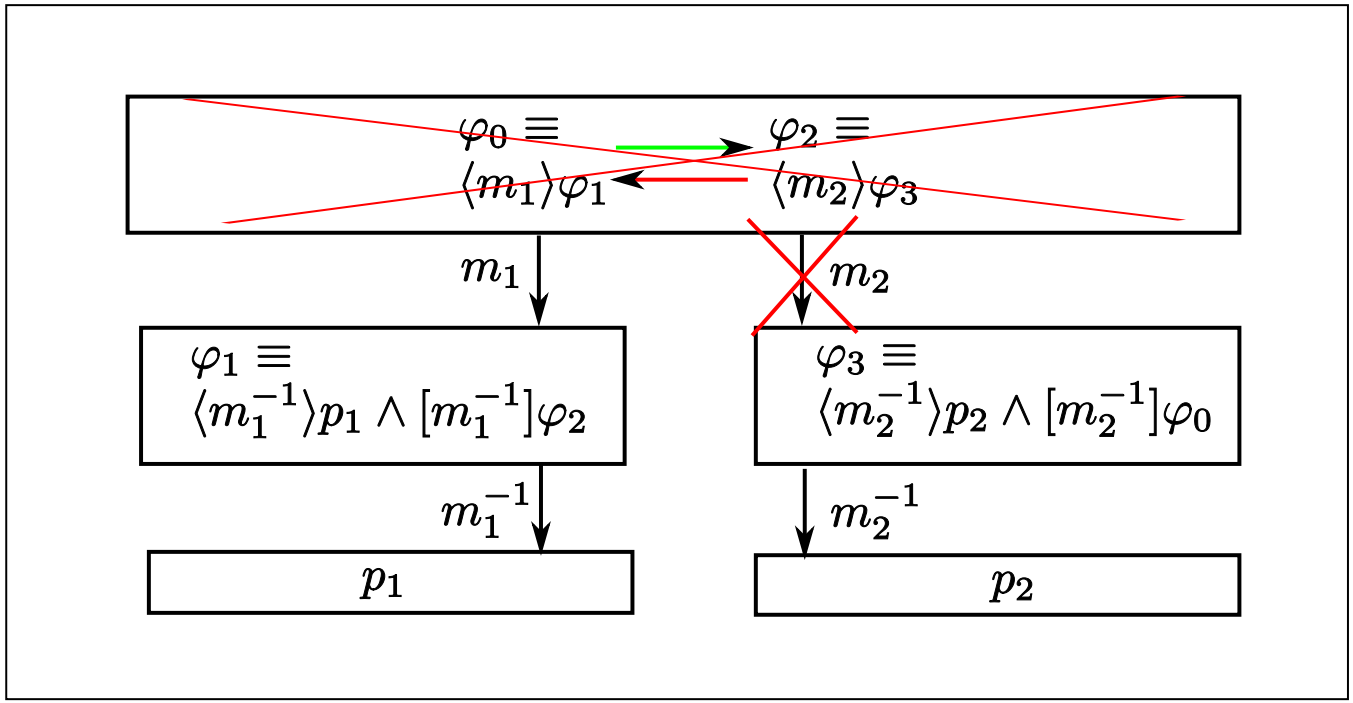
Backward Modality: Solution to Problem 2



Backward Modality: Solution to Problem 2



Backward Modality: Solution to Problem 2



Nominals: pseudo-nominal

If x is nominal,

- $9s \in S$: $K; s \models x$
- $K; s \models x \;\&\; K: s' \models x \Rightarrow s = s'$.

(Propositional symbol) x is a pseudo-nominal in K , if:

- $9s \in S$: $K; s \models x$
- $K; s \models x \;\&\; K: s' \models x \Rightarrow 8 \in cl(\ ): K; s \models \ () \ K \ ; s' \models$

Nominals: pseudo-nominal

g : Nom ! P (cl(' |)) is a naming function, if

- x 2 g(x)
- x 2 g(x^0) =) g(x) = g(x^0)

Used already in [Sattler, Vardi 01]

If x is a pseudo-nominal in K ,

$$g : x \mapsto f\ ' \ 2 \mathrm{cl}(' \mid) \mid L^0(x) \not\models ' \ g$$

is (well-de ned and) a naming function.

Nominals: pseudo-nominal

Outline of the procedure for given formula ' :

1. Guess a naming function g.
2. Start with all possible nodes.
3. Remove all inconsistent nodes.
   - : -consistency, -consistency, and -consistency.
   - Node n such that x 2 n and n , g(x).
4. Repeat the previous step until all remaining nodes are consistent.
5. If there is a node that contains ' , ' is satis able.
6. Otherwise, Go back to the beginning and try another g. Repeat this process until all g has been checked.

ominals: duplicated node

Nominals: avoiding duplication

Assume $K ; s \models ' $ and $' $ is a -formula. Let $\bar{y}(s; ')$ be the number of nominals contained in the witness DAG D for $(s; ')$.

- If $(s^0; ' ^0)$ is a successor of $(s; ')$ in D, then $\bar{y}(s^0; ' ^0) \quad \bar{y}(s; ')$.
- Furthermore, if $s^0 \models x$ for some $x \in \mathrm{Nom}$, then $\bar{y}(s^0; ' ^0) < \bar{y}(s; ')$.

Now, each node n in the Tab is in the form of $n = (; x; y)$.
y carries the information of $\bar{y}$.

Correctness

' $_{\mathrm{I}}$ is satis able $()^{?}$ the procedure returns 'yes'

-) holds if ' $_{\mathrm{I}}$ is in L ($_{\mathrm{AF}}$; nom; back; func)
- (holds if ' $_{\mathrm{I}}$ is in L ($_{\mathrm{AF}}$; back; func), L ($_{\mathrm{AF}}$; nom; func), or L ($_{\mathrm{AF}}$; nom; back),

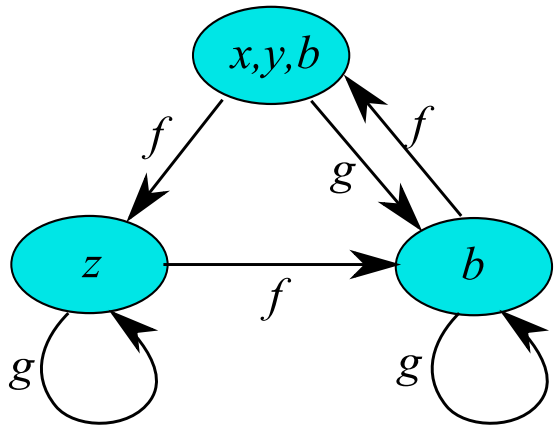
Shape analysis

- Shape analysis: Analysis of programs that manipulates pointers.
- We regard the heap as a Kripke structure.

Example:

```
struct Node {
    Node* f;
    Node* g;
    boolean b;
}

Node* x,y,z;
```



PS = fbg
boolean elds

FMS = ff ; gg
pointer elds

Nom = fx; y; zg
pointer type variables

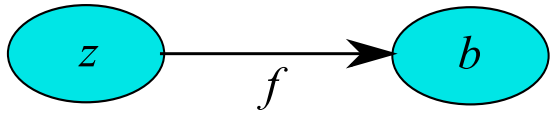
Properties of the heap

Properties of the heap is expressed by formulas.

Example:

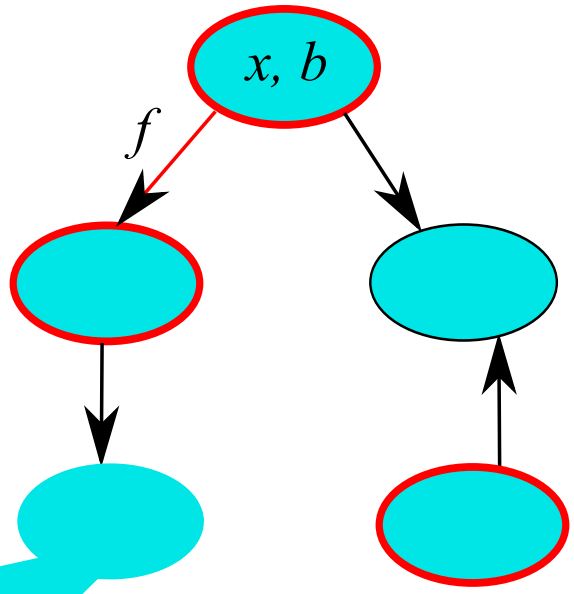
@zhfib

The f-successor of z satis es b.



@x X(y _ (b^hfiX) _ (: b^hgiX))

y is reachable from x by following pointer f or g, depending on whether b is satis ed or not.



Hoare-style proof example

Judging Hoare triples

Hoare triple for “basic operation” can be veri ed using the w eakest precondition and a decision procedure

If a decision procedure judges that

$$' \quad ! \quad \mathrm{wp}(\ ;\ \ ) \quad \text{is valid,}$$

then hoare triple

$$f' \ g \ \ f \ \ g \quad \text{is correct.}$$

For this purpose, the decision procedure must be sound, but need not to be complete.

Implementation

A system that helps the user to build Hoare-style proofs is implemented:

- Written in Java.
- Uses BDD (JavaBDD) for set operations.
- Pseudo-nominality.
- L ($_{AF}$; nom; func) with strongest postconditions.

Performance Examples

id	len	# cl	# nom	time (ms)
regr56	35	11	3	461
listRevNoLeakB	92	25	5	610
listRevSwapA	148	37	6	1199
dswPopC2	188	46	7	2615
dswPushA1	372	54	7	9469

Pentium 4 CPU 2.4GHz, 1GB mem, Windows XP.

Conclusion

Summary

- An implementable decision procedure for variants of enriched modal -calculus.
- Application to program analysis.

Future Work

- Implementation of the full decision procedure.
- Decision procedures for other variants, such as:
  - -GF, guarded fragments of the rst order logic with xed-poin t operators.
  - Some variants of multi-valued -calculus.

Thank you for your attention.

reg56 =

```
  @ x1 nu Y1 ( !b1 & [n1] Y1 )
& @ x2 nu Y1 ( !b1 & [n1] Y1 )
& @ x1 mu X1 ( <n1> X1 | b1 )
& @ NULL <n1> NULL
```

dswPushA1 =

```
( ( @ p ( p | ( <left> t & !p ) )
    & @ p <left> x
    & @ p mu XC ( ( ( @ t XC & p )
                  | ( !p & <left> XC )
                  | <right> XC
                  | a
                  )
                & ( !marked | p )
                )
    & @ p ( !marked | p )
    & @ p !NULL
    & @ p marked
    & @ x mu X0 ( ( ( ( <right> X0 & swung )
                    | ( !swung
                      & ( ( @ t X0 & p )
                        | ( !p & <left> X0 )
                        )
                      )
                    )
                  & marked
                  & !p
                  )
                | NULL
                )
    & @ a ( ( ( <left> b & !p )
            | ( p & @ t b )
            )
          & <right> c
          & ( !marked | p )
          )
    & @ a !NULL
    & @ a !p
    )
| ( @ p ( p | ( <left> t & !p ) )
    & @ p <left> x
    & @ p ( !marked | p )
    & @ p mu XC ( ( ( @ t XC & p )
                  | ( !p & <left> XC )
                  | <right> XC
                  | a
                  )
                & ( !marked | p )
                )
    & @ x mu X0 ( ( marked
                  & ( ( !swung
                      & ( ( @ t X0 & p )
                        | ( !p & <left> X0 )
                        )
                      & !p
                      )
                    | ( ( swung | p ) & <right> X0 )
                    )
                  & !p
                  )
                | NULL
                )
    & @ p !NULL
    & @ p marked
    & @ a ( ( ( <left> b & !p )
            | ( p & @ t b )
            )
          & <right> c
          & ( !marked | p )
          )
    & @ a !p
    & @ a !NULL
    )
  )
& @ NULL <left> NULL
& @ p !swung
& ( ( ( @ p nu XE ( ( !swung | [right] XE )
                  & [right] nu XC ( ( [right] XC
                                    & [left] XC
                                    & !a
                                    )
```

```
                                        | marked
                                        )
                    & ( [left] XE | swung )
                    )
      | @ a t
      | @ a ( [left] !b
               | marked
               | [right] !c
               )
    )
  & ( @ a ( [left] !b
               | marked
               | [right] !c
               )
      | @ a !t
    )
  & ( @ t nu XC ( ( [right] XC
                    & [left] XC
                    & !a
                    )
                  | marked
                  )
      | @ a t
      | @ a ( [left] !b
               | marked
               | [right] !c
               )
    )
  )
| @ p nu X0 ( ( ( ( !swung | [right] X0 )
                  & ( swung | [left] X0 )
                  )
                | !marked
                )
              & !NULL
              )
| @ a NULL
)
& @ NULL <right> NULL
```