Nested Words and Trees

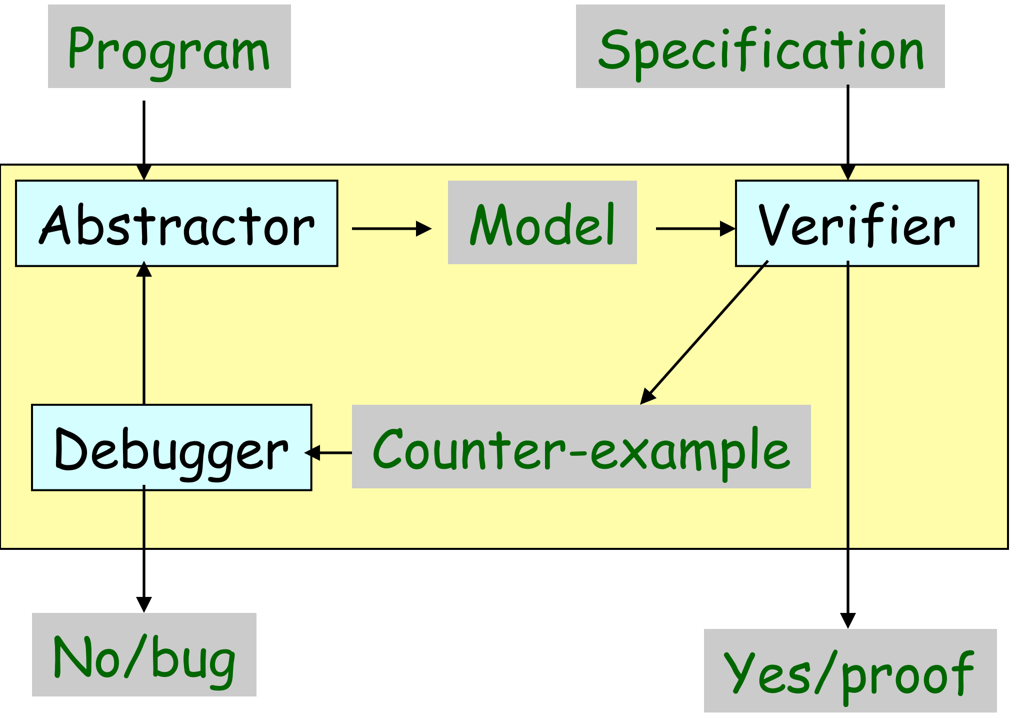
Rajeev Alur

University of Pennsylvania

Joint work with S.Chaudhuri & P.Madhusudan

Games Workshop, Cambridge, UK, July 2006

Software Model Checking



Research challenges

- Search algorithms
- Abstraction
- Static analysis
- Refinement
- Expressive specs

Applications

- Device drivers, OS code
- Network protocols
- Concurrent data types

Tools: SLAM, Blast, CBMC, F-SOFT

Classical Model Checking

- Both model M and specification S define regular languages
  - M as a generator of all possible behaviors
  - S as an acceptor of "good" behaviors (verification is language inclusion of M in S) or as an acceptor of "bad" behaviors (verification is checking emptiness of intersection of M and S)
- Typical specifications (using automata or temporal logic)
  - Safety: Lock and unlock operations alternate
  - Liveness: Every request has an eventual response
  - Branching: Initial state is always reachable
- Robust foundations
  - Finite automata / regular languages
  - Buchi automata / omega-regular languages
  - Tree automata / parity games / regular tree languages

Checking Structured Programs

- Control-flow requires stack, so model M defines a context-free language
- Algorithms exist for checking regular specifications against context-free models
  - Emptiness of pushdown automata is solvable
  - Product of a regular language and a context-free language is context-free
- But, checking context-free spec against a context-free model is undecidable!
  - Context-free languages are not closed under intersection
  - Inclusion as well as emptiness of intersection undecidable
- Existing software model checkers: pushdown models (Boolean programs) and regular specifications

Are Context-free Specs Interesting?

- Classical Hoare-style pre/post conditions
  - If p holds when procedure A is invoked, q holds upon return
  - Total correctness: every invocation of A terminates
  - Integral part of emerging standard JML
- Stack inspection properties (security/access control)
  - If setuuid bit is being set, root must be in call stack
- Interprocedural data-flow analysis
- All these need matching of calls with returns, or finding unmatched calls
  - Recall: Language of words over [,] such that brackets are well matched is not regular, but context-free

Checking Context-free Specs

- Many tools exist for checking specific properties
  - Security research on stack inspection properties
  - Annotating programs with asserts and local variables
  - Inter-procedural data-flow analysis algorithms

- What's common to checkable properties?
  - Both model M and spec S have their own stacks, but the two stacks are synchronized

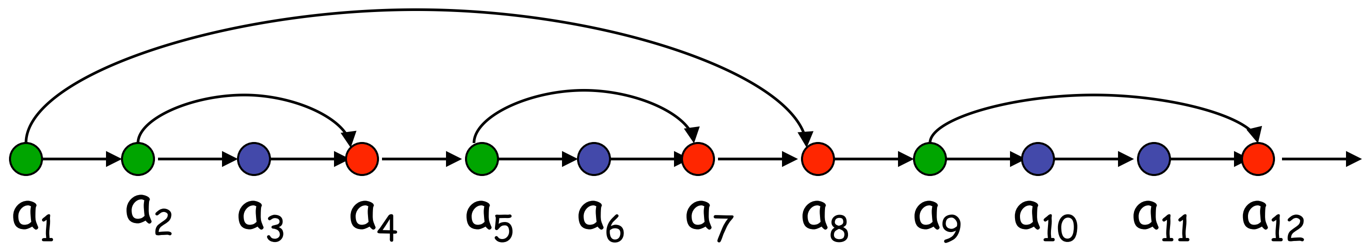
- As a generator, program should expose the matching structure of calls and returns

Solution: Nested words and theory of regular languages over nested words

Nested Words

Nested word:

- Linear sequence + well-nested edges
- Positions labeled with symbols in S



Positions classified as:

- Call positions: both linear and hierarchical successors
- Return positions: both linear and hierarchical predecessors
- Internal positions: otherwise

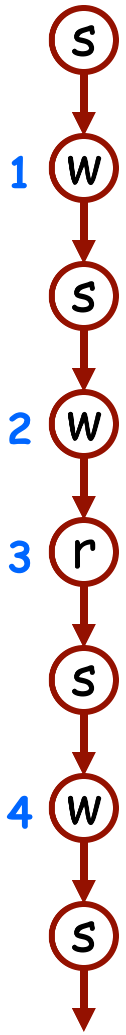
Assume each position has at most one nested edge

Program Executions as Nested Words

Program

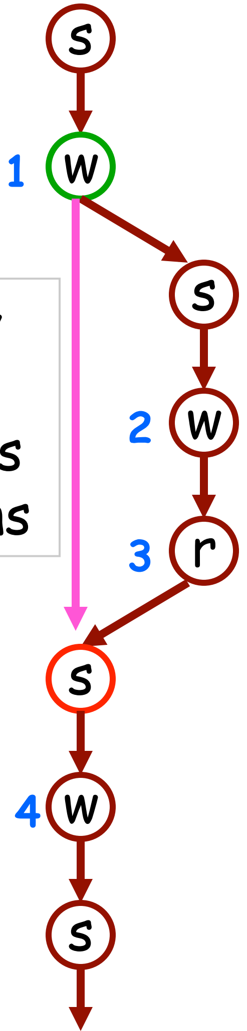
```
bool P() {
  local int x,y;
  …
  x = 3; 1
  if Q   x = y ; 4
  …
}

bool Q () {
  local int x;
  …
  x = 1; 2           3
  return (x==0);
}
```

An execution as a word



Symbols:
w : write x
r : read x
s : other

An execution as a nested word



Summary edges from calls to returns

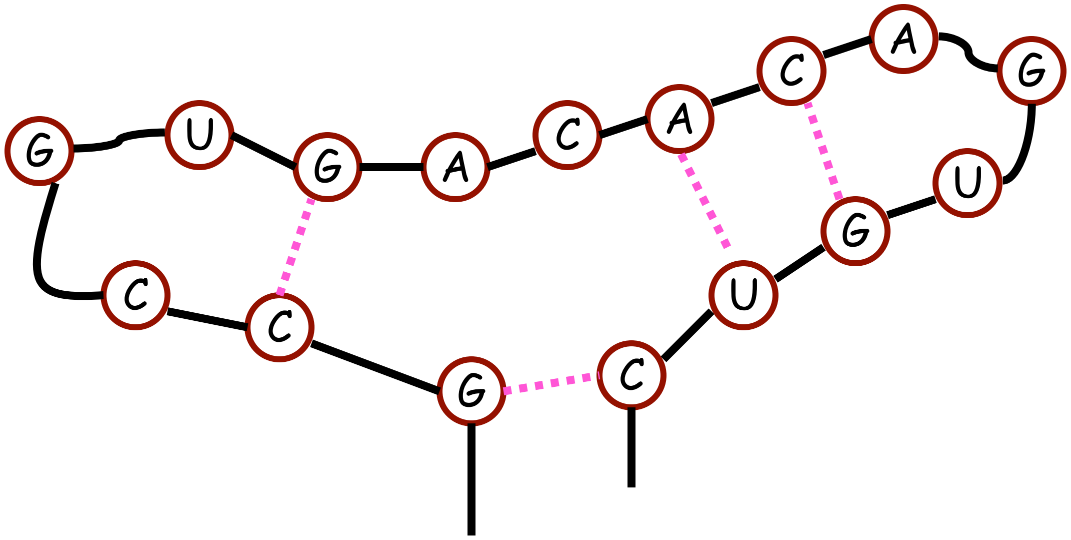
Model for Linear Hierarchical Data

- Nested words: both linear and hierarchical structure is made explicit. This seems natural in many applications
  - Executions of structured program
  - RNA: primary backbone is linear, secondary bonds are well-nested
  - XML documents: matching of open/close tags
- Words: only linear structure is explicit
  - Pushdown automata add/discover hierarchical structure
  - Parantheses languages: implicit nesting edges
- Ordered Trees: only hierarchical structure is explicit
  - Ordering of siblings imparts explicit partial order
  - Linear order is implicit, and can be recovered by infix traversal

RNA as a Nested Word

Primary structure: Linear sequence of nucleotides (A, C, G, U)

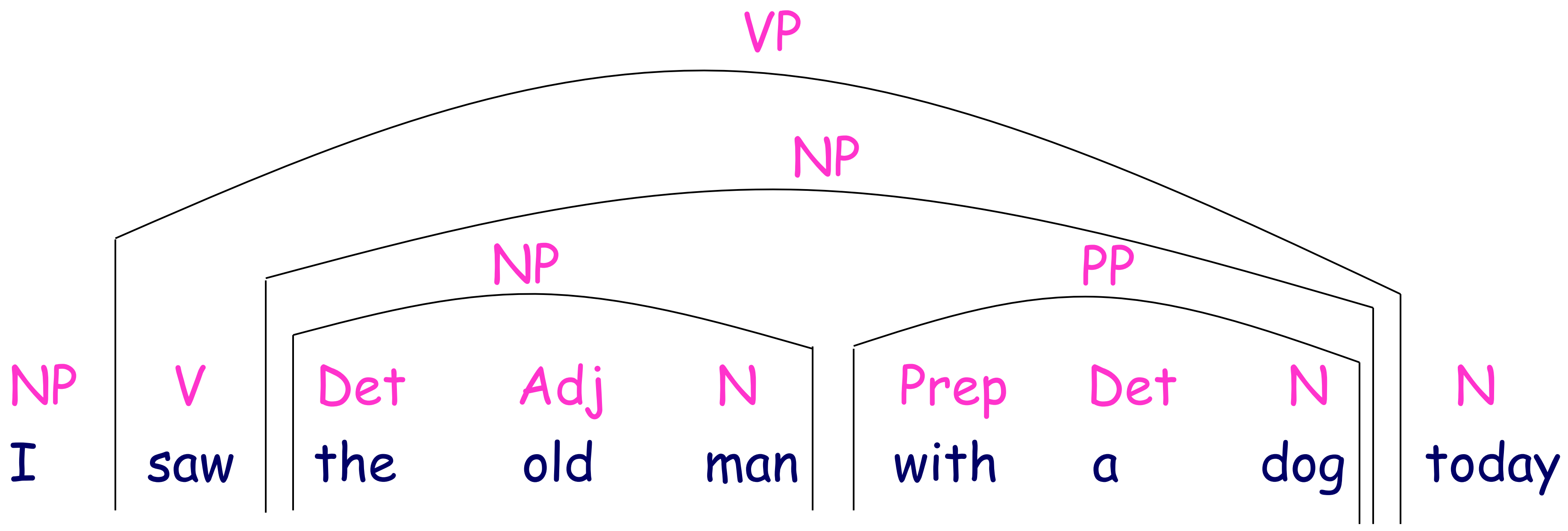
Secondary structure: Hydrogen bonds between complementary nucleotides (A-U, G-C, G-U)



In literature, this is modeled as trees.

Algorithmic question: Find similarity between RNAs using edit distances

Linguistic Annotated Data


VP
NP
NP
PP
NP V Det Adj N Prep Det N N
I saw the old man with a dog today

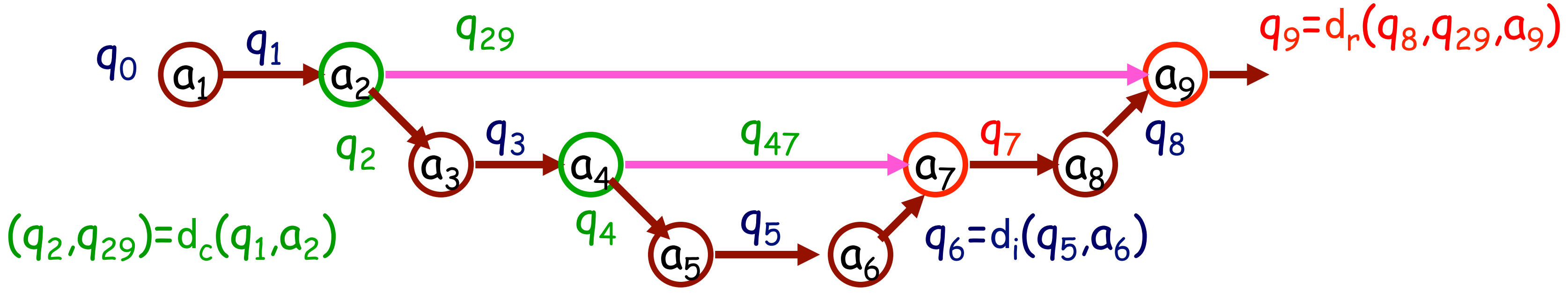
Linguistic data stored as annotated sentences (eg. Penn Treebank)

Nested words, possibly with labels on edges

Sample query: Find nouns that follow a verb which is a child of a verb phrase

Existing query languages: XPath, XQuery, LPath (BCDLZ)

Nested Word Automata (NWA)



q_0
q_1
q_{29}
$q_9=d_r(q_8,q_{29},a_9)$
q_2
q_3
q_{47}
q_7
q_8
$(q_2,q_{29})=d_c(q_1,a_2)$
q_4
q_5
$q_6=d_i(q_5,a_6)$
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9

- States Q, initial state q_0, final states F
- Starts in initial state, reads the word from left to right labeling edges with states, where states on the outgoing edges are determined from states of incoming edges
- Transition function:
  - d_c : Q x S -> Q x Q (for call positions)
  - d_i : Q x S -> Q (for internal positions)
  - d_r : Q x Q x S -> Q (for return positions)
- Nested word is accepted if the run ends in a final state

Regular Languages of Nested Words

- A set of nested words is regular if there is a finite-state NWA that accepts it

- Nondeterministic automata over nested words
  - Transition function: d_c: $Q \times S \to 2^{Q \times Q}$, d_i : $Q \times S \to 2^Q$, d_r: $Q \times Q \times S \to 2^Q$
  - Can be determinized

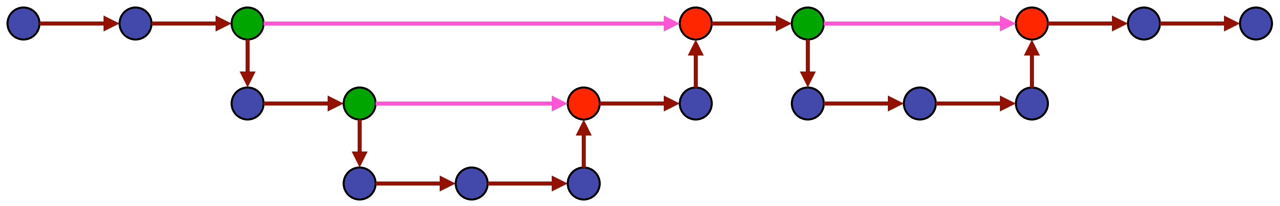
- Graph automata over nested words defined using tiling systems are equally expressive (edges out of a call position have separate states)

- Appealing theoretical properties
  - Effectively closed under various operations (union, intersection, complement, concatenation, projection, Kleene-* …)
  - Decidable decision problems: membership, language inclusion, language equivalence …
  - Alternate characterization: MSO, syntactic congruences

Application: Software Analysis

- A program *P* with stack-based control is modeled by a set *L* of nested words it generates
  - Choice of S depends on the intended application
  - Summary edges exposing call/return structure are added (exposure can depend on what needs to be checked)
  - If *P* has finite data (e.g. pushdown automata, Boolean programs, recursive state machines) then *L* is regular
- Specification *S* given as a regular language of nested words
- Verification: Does every behavior in *L* satisfy *S* ?
  - Runtime monitoring: Check if current execution is accepted by S (compiled as a deterministic automaton)
  - Model checking: Check if *L* is contained in *S*, decidable when *P* has finite data

Writing Program Specifications



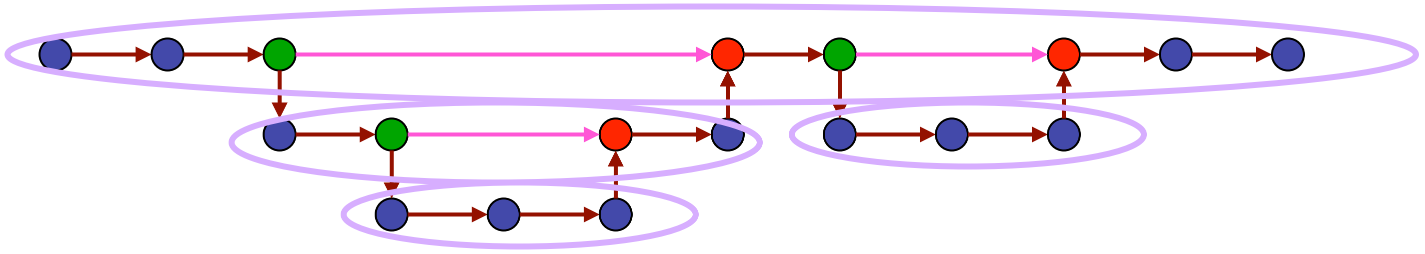
Intuition: Keeping track of context is easy; just skip using a summary edge

- Finite-state properties of paths, where a path can be a local path, a global path, or a mixture

Sample regular properties:

- If *p* holds at a call, *q* should hold at matching return
- If *x* is being written, procedure *P* must be in call stack
- Within a procedure, an *unlock* must follow a *lock*
- All properties specifiable in standard temporal logics (LTL)

Local Regularity



Let L be a regular language,

Local(L): every local path is in L (skip summary edges)

- E.g. L: every write (w) is followed by a read (r)

Given a DFA A for L, construct NWA B for Local(L)

- States Q, initial state q_0, final states F, same as A
- $d_i(q,a) = d(q,a)$
- $d_c(q,a) = (q_0, d(q,a))$
- $d_r(q,q',a) = d(q',a)$ if q is in F

Application: Document Processing

XML Document

```
<conference>
 <name>
    DLT 2006
 </name>
 <location>
    <city>
      Santa Barbara
    </city>
    <hotel>
      Best Western
    </hotel>
  </location>
  <sponsor>
     UCSB
  </sponsor>
  <sponsor>
     Google
  </sponsor>
</conference>
```

Query Processing

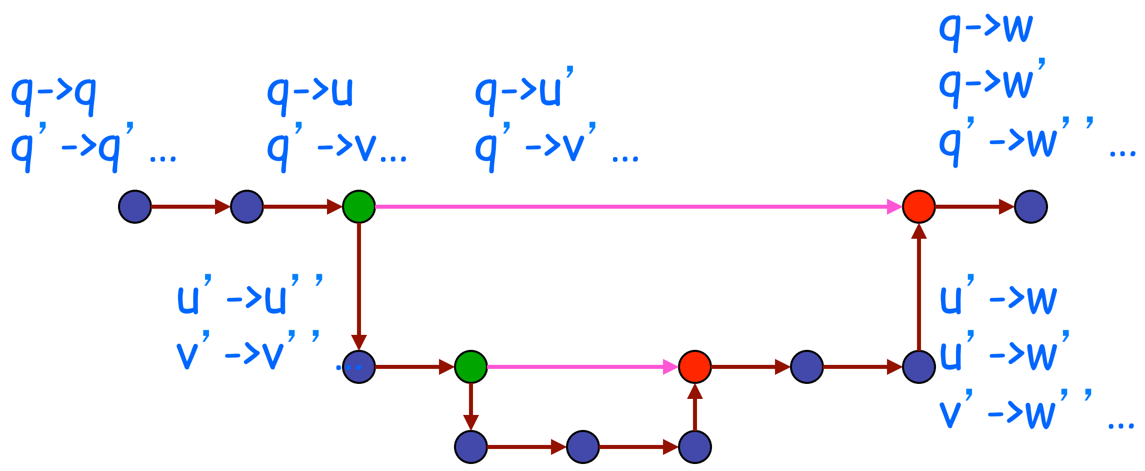
Model a document *d* as a nested word
   Nesting edges from <tag> to </tag>

Sample Query: Find documents related to conferences sponsored by Google in Santa Barbara

Specify query as a regular language *L* of
  nested words
Analysis: Membership question
   Does document *d* satisfy query *L* ?

Use NWA instead of tree automata!
(typically, no recursion, but only hierarchy)
Useful for streaming applications, and when data has also a natural linear order

Determinization



Goal: Given a nondeterministic automaton A with states Q, construct an equivalent deterministic automaton B

- Intuition: Maintain a set of "summaries" (pairs of states)
- State-space of B: $2^{Q\times Q}$
- Initially, state contains q->q, for each q
- At call, if state u splits into (u' ,u''), summary q->u splits into (q->u' ,u' ->u'')
- At return, summaries q->u' and u' ->w join to give q->u
- Acceptance: must contain q->q' , where q is initial and q' is final

Closure Properties

The class of regular languages of nested words is effectively closed under many operations

- Intersection: Take product of automata (key: nesting given by input)
- Union: Use nondeterminism
- Complementation: Complement final states of deterministic NWA
- Projection: Use nondeterminism
- Concatenation/Kleene*: Guess the split (as in case of word automata)
- Reverse (reversal of a nested word reverses nested edges also)

Decision Problems

- Membership: Is a given nested word w accepted by NWA *A*?
  - Solvable in polynomial time
  - If *A* is fixed, then in time O(|w|) and space O(nesting depth of w)

- Emptiness: Given NWA *A*, is its language empty?

  Solvable in time $O(|A|^3)$: view *A* as a pushdown automaton

- Universality, Language inclusion, Language equivalence:
  - Solvable in polynomial-time for deterministic automata
  - For nondeterministic automata, use determinization and complementation; causes exponential blow-up, Exptime-complete problems

MSO-based Characterization

- Monadic Second Order Logic of Nested Words
  - First order variables: x,y,z; Set variables: X,Y,Z…
  - Atomic formulas: a(x), X(x), x=y, x < y, x -> y
  - Logical connectives and quantifiers

- Sample formula:

  For all x,y. ((a(x) and x -> y) implies b(y))

  Every call labeled *a* is matched by a return labeled *b*

- Thm: A language L of nested words is regular iff it is definable by an MSO sentence
  - Robust characterization of regularity as in case of languages of words and languages of trees

MSO-NWA Equivalence (Proof sketch)

- From deterministic NWA to MSO
  - Unary predicates and q_l and q_h for each state q of *A*
  - Formula says that these predicates encode a run of *A* consistent with its transition function (q_h is used to encode state-labels on nesting edges)
  - d_r requirement can be encoded using nesting-edge predicate $\rightarrow$

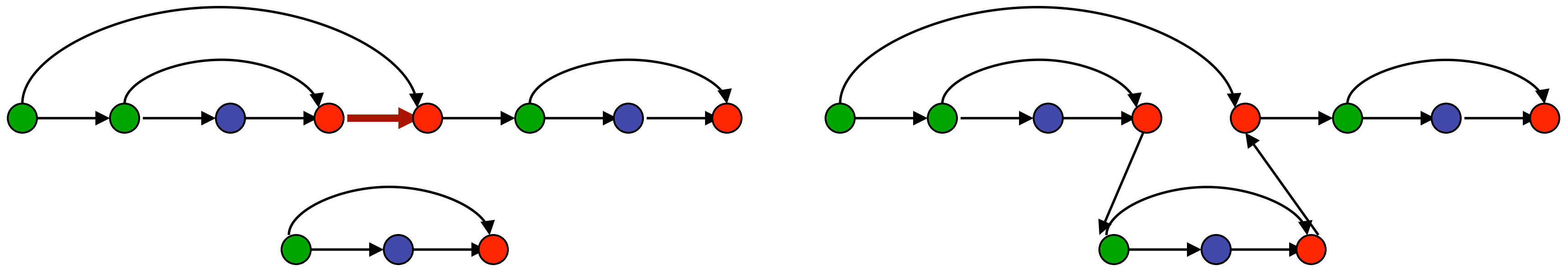
- Only existential-second-order prefix suffices

- From MSO to nondeterministic NWA
  - NWA can check base predicates $x=y$, $x < y$, $x \rightarrow y$
  - Use closure properties: union, complement, and projection

Congruence Based Characterization

Context *C*: A nested word and a linear edge

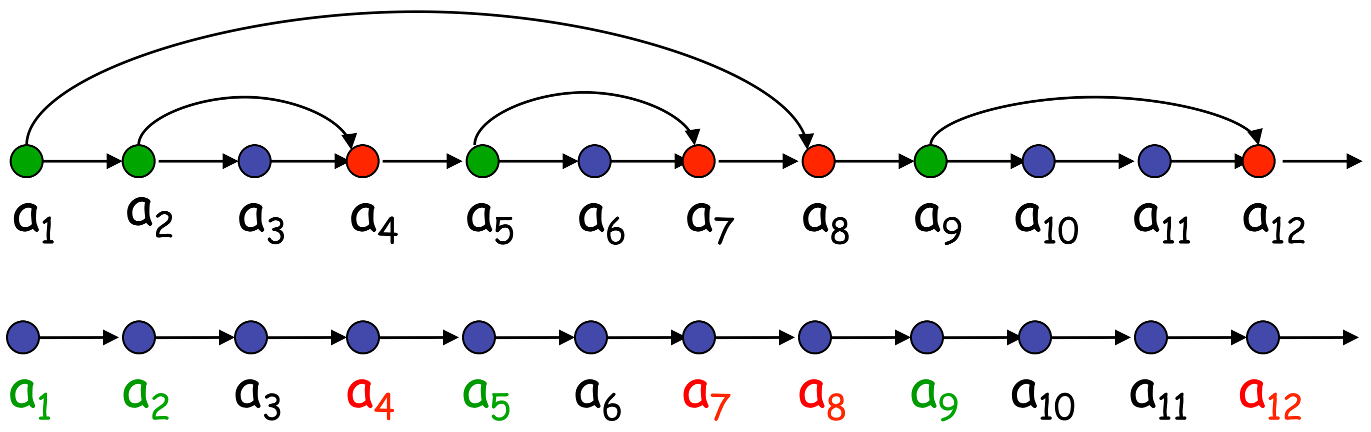
Substitution I(*C*,w): Insert nested word w in a context *C*



Congruence: Given a language *L* of nested words, $w \sim_L w'$ if for every context *C*, I(*C*,w) is in *L* iff I(*C*,w') is in *L*

Thm: A language *L* of nested words is regular iff the congruence $\sim_L$ is of finite index.

Relating to Word Languages



a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12}
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12}

Words labeled with a typed alphabet (visibly pushdown words)

- Symbols partitioned into calls, returns, and internals
- Two views are basically the same giving similar results

Visibly Pushdown Automata

- Pushdown automaton that must push while reading a call, must pop while reading a return, and not update stack on internals
- Height of stack determined by input word read so far

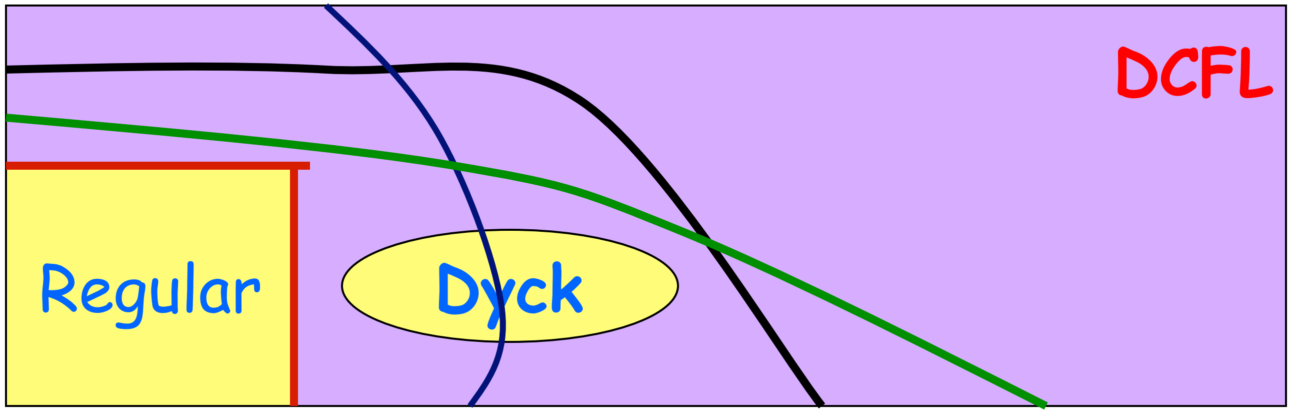
Visibly Pushdown Languages

- A robust subclass of deterministic context-free languages

VPLs vs DCFLs

Fix Σ. For each partitioning of Σ into Σ_c, Σ_i, Σ_r, we get a corresponding class of visibly pushdown languages

- Each class is closed under Boolean operations
- Decidable equivalence, inclusion problems etc

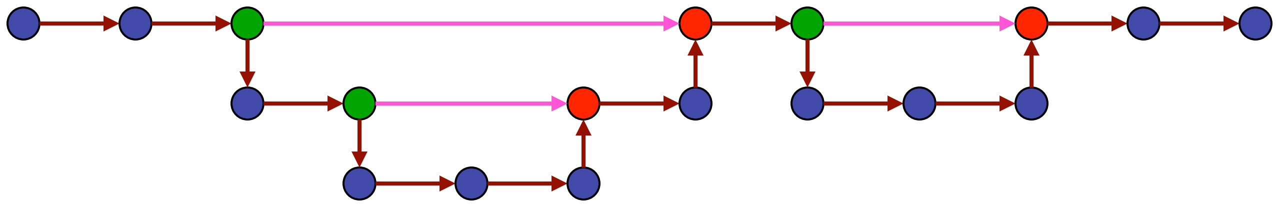


Are these VPLs?

$L_1 = \{a^n b^n \mid n > 0\}$, $L_2 = \{b^n a^n \mid n > 0\}$, L_3 = words with same # of a's & b's

Instead of static typing of symbols, one can use dynamic types determined by an automaton to get more VPL classes[Caucal'06]

Relating to Tree Languages



A binary tree is hiding in a nested word

- At calls, left subtree encodes what happens in the called procedure, and right subtree gives what happens after return

Why not use tree encoding and tree automata ?

- Notion of regularity is same in both views
- Nesting is encoded, but linear structure is lost
- Deterministic tree automata are not expressive
- No notion of reading input from left to right
- XML literature has lots of attempts to address this deficiency: Tree walking automata…

Summary Table

	Word Automata	Pushdown Automata	Tree Automata	NWA
Union	yes	yes	yes	yes
Intersection	yes	no	yes	yes
Complement	yes	no	yes	yes
Det= Nondet	yes	no	no	yes
Emptiness	Nlogspace	Ptime	Ptime	Ptime
Inclusion (Nondet)	Pspace	Undec	Exptime	Exptime

Related Work

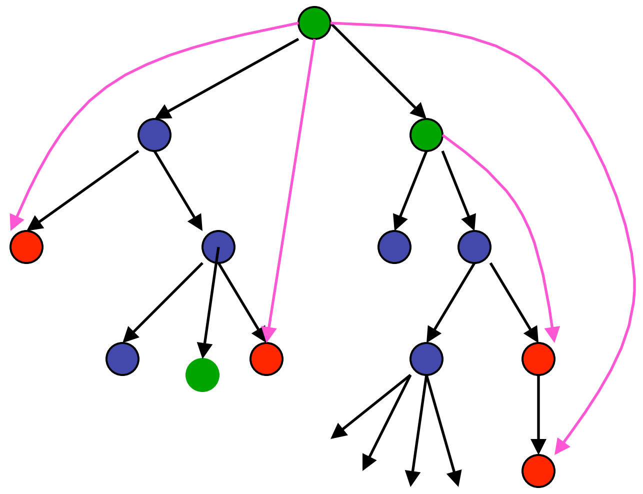
- Restricted context-free languages
  - Parantheses languages, Dyck languages
  - Input-driven languages
- Logical characterization of context-free languages (LST'94)
- Connection between pushdown automata and tree automata
  - Set of parse trees of a CFG is a regular tree language
  - Pushdown automata for query processing in XML
- Algorithms for pushdown automata compute summaries
  - Context-free reachability
  - Inter-procedural data-flow analysis
- Model checking of pushdown automata
  - LTL, CTL, m-calculus, pushdown games
  - LTL with regular valuations of stack contents
  - CaRet (LTL with calls and returns)

Research Directions

- Visible Pushdown Languages (AM, STOC'04)
  - Extends to w-regular languages of infinite words
- VPL triggered research
  - Games (LMS, FSTTCS' 04)
  - Congruences and minimization (AKMV ICALP' 05, KMV Concur' 06)
  - Third-order Algol with iteration (MW FoSSaCS' 05)
  - Dynamic logic with recursive programs (LS FoSSaCS' 06)
  - Synchronization of pushdown automata (Caucal DLT' 06)
- Linear-time Temporal Logics
  - CaRet (Logic of calls and returns) (AEM TACAS' 04)

Caution: Not studied in the nested word framework

Nested Trees



Tree edges + Nesting edges

Unranked (arity not fixed)

Unordered

Infinite

Given a pushdown automaton (or a Boolean program) A, model it by a nested tree T_A

- Each path models an execution as a nested word
- Branching-time model checking: Specification is a language of nested trees, verification is membership

Tree Automata Definitions

Transition function of a tree automaton d : Q x S -> D

D depends on type of automaton and type of trees

- ❑ Nondeterministic over binary trees: D is a set of pairs; A choice (u,v) means send u to left child and v to right child

- ❑ Nondeterministic over ordered trees: D is a regular language over Q: the sequence of states sent along children must be in D

- ❑ Nondeterministic over unordered unranked trees: D is a set of terms in 2^Q x Q; A choice $(\{q_1,q_2\}, q_3)$ means that send q_1 to one child, q_2 to a different child, and q_3 to all remaining children

- ❑ Alternating over unordered unranked trees: D contains formulas that positive Boolean combination of terms of the form <q>, [q]; A formula $(<q_1>$ or $<q_2>)$ and $[q_3]$ means send q_3 to all children, and either q_1 or q_2 to one of them

Nondeterministic Nested Tree Automata

- Finitely many states Q, initial states

- Run of the automaton: Labeling of edges with states consistent with initial set and transition function

- Local transitions: $d_i(q,a)$ is a set of terms in $2^Q \times Q$

- Call transitions: $d_c(q,a)$ is a set of terms in $2^{Q \times Q} \times Q \times Q$; $(\{(q_1,q_2)\},q_3,q_4)$ means send q_1 to one child, q_2 along corresponding nesting edges, q_3 to remaining children, and q_4 along all remaining nesting edges

- Return transitions: $d_r(q,q',a)$ is set of terms in $2^Q \times Q$, here q is the state along tree edge, and q' is the state along nesting edge

- Acceptance condition: Final states, Buchi, Parity (NPNTA)

Properties of NPNTAs

- **Thm:** Closed under union and projection.
- **Thm:** Closed under intersection.
  Proof idea: Finite-state; just take product.
- **Thm:** Not closed under complement.
- **Thm:** Emptiness checkable in EXPTIME.
  Proof idea: Special case of empti
  automata.
- **Thm:** Model-checking on pushd
  Proof idea: The stack of the in
  synchronized with the implicit s
  construction works.
- **Thm:** Universality undecidable.

Extension: alternation. Extra expressive power, unlike in the case of tree automata

Alternating Nested Tree Automata

- Transition Terms (TT): Positive Boolean combination of atomic terms of the form <q> (send q to some child), [q] (send q to all children)

- CTT: Positive boolean combination of terms of the form
  <q,q'> (send q to some child and q' to all corresponding nesting edges)
  [q,q'] (q on all tree edges, q' on all nesting edges)

- Transition function has call, return and internal components:
  d_i : Q x S -> TT, d_c : Q x S -> CTT, d_r : Q x Q x S -> TT

- Run of the automaton: game between the automaton and an adversary.

- Winning condition: Parity

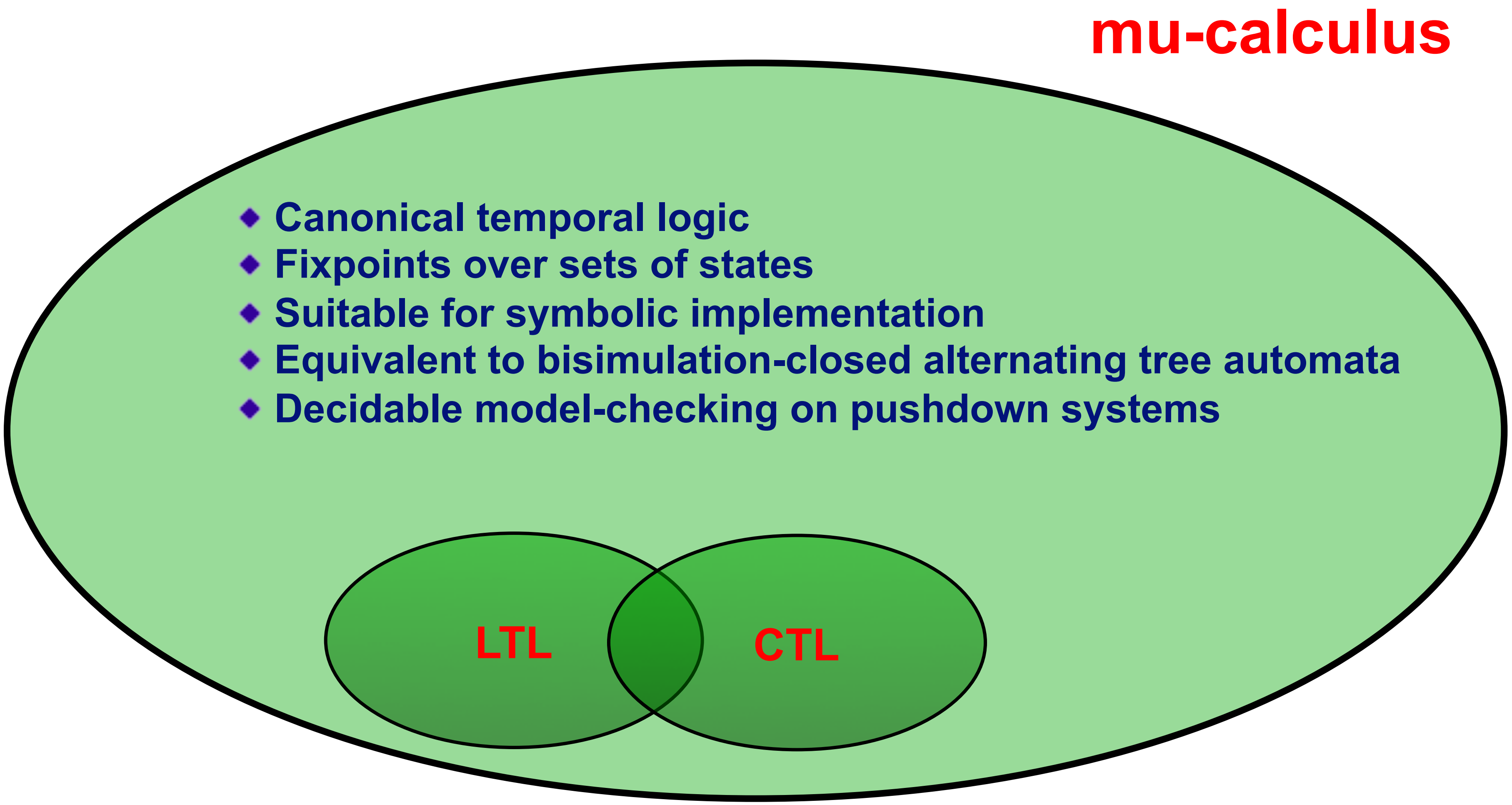
- Tree accepted iff automaton has a winning strategy

Properties of APNTAs

- **Thm:** Closed under union, intersection.
- **Thm:** Closed under complement.
  Proof idea: Parity games are determined, and designing the complement game is easy.
- **Thm:** Not closed under projection.
- **Thm:** Can express some non-context-free tree languages.
- **Theorem:** Model-checking EXPTIME-complete
  Proof idea: Stack of the inpu
  with the implicit stack of the
  to a pushdown game, solvable
- **Thm:** Emptiness, universality

Next…
Logics on nested trees

Logics for Trees

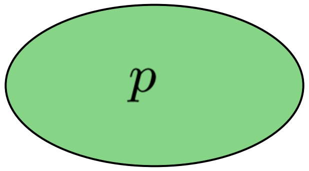


- Canonical temporal logic
- Fixpoints over sets of states
- Suitable for symbolic implementation
- Equivalent to bisimulation-closed alternating tree automata
- Decidable model-checking on pushdown systems

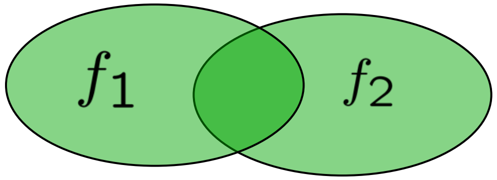
$$\mu X.(p \vee \langle\rangle X)$$

Mu-Calculus

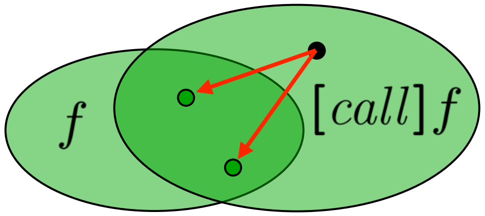
Assembly language of temporal logics
Formulas → Sets of nodes



$$p \qquad \neg p$$



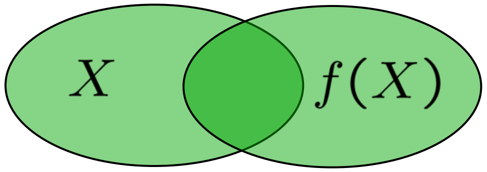
$$f_1 \wedge f_2 \qquad f_1 \vee f_2$$



$$\langle call \rangle f \qquad [call] f$$

$$X$$

$f(X)$: Node set $\mapsto$ Node set



$$\mu X.f(X) \qquad \nu X.f(X)$$

Least and greatest fixpoints of *f*

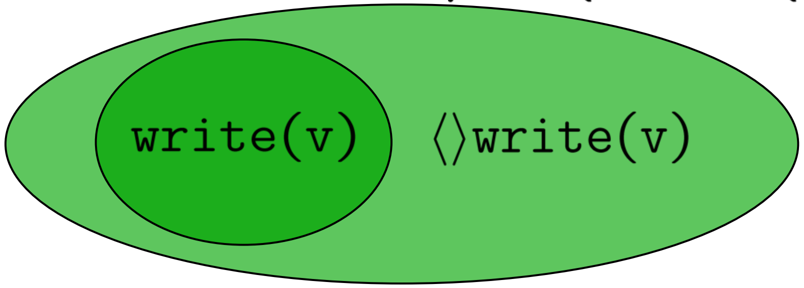
<call>f, <ret>f, <loc>f : there is an edge to call/ret/local node satisfying f

Fixpoints in mu-calculus

Model-checking mu-calculus on pushdown systems is decidable. But…

Reachability in mu-calculus:

$$\mu X.(\mathtt{write(v)} \vee \langle\rangle \mathtt{X})$$

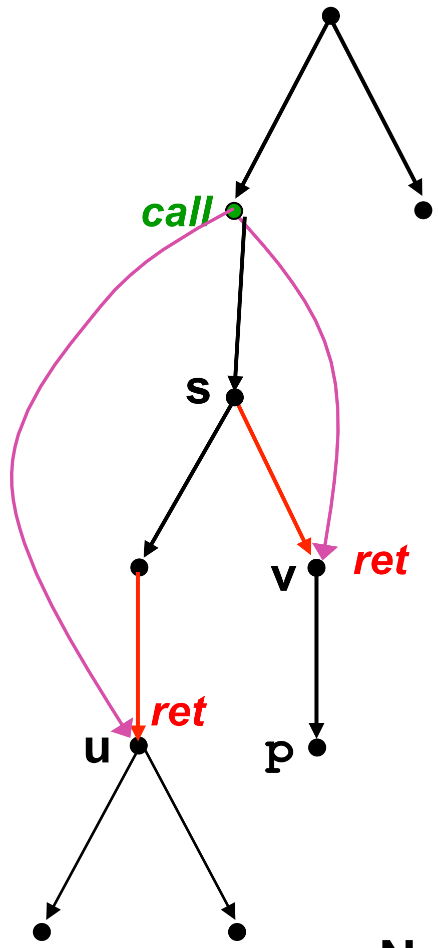


Formula describes a terminating symbolic computation for finite-state systems.

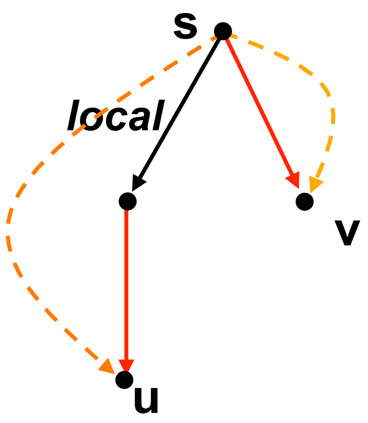
Application: mu-calculus is the "assembly language" in temporal logic model-checkers like NuSMV.

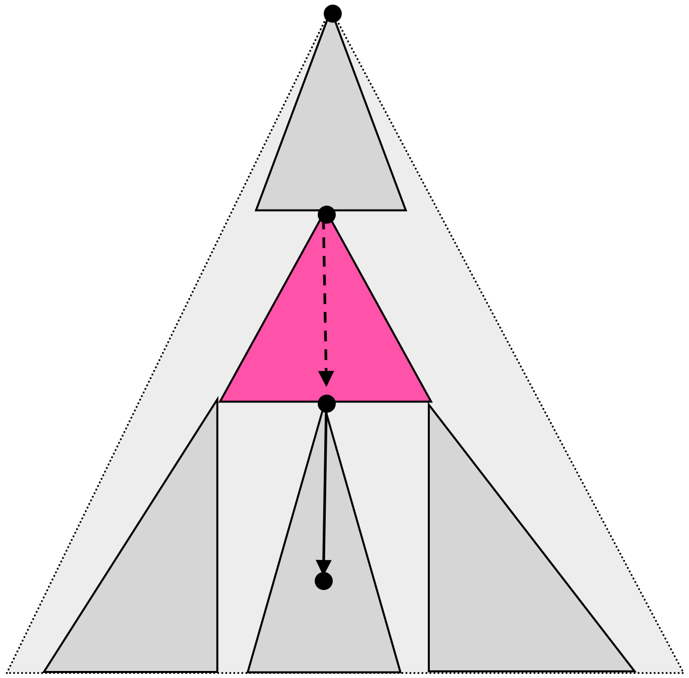
What about pushdown models (interprocedural analysis)? Algorithms use "summarization", and not captured by mu-calculus

Summary Subtrees



Summary

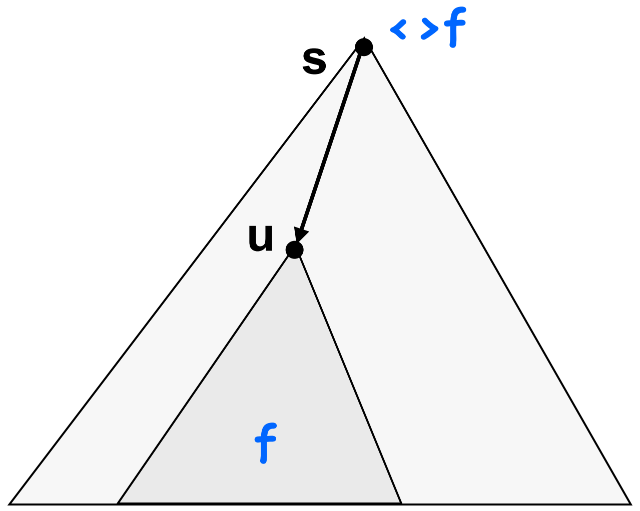




Matching returns of s = {u,v}

Nesting edges let us chop a nested tree into subtrees that *summarize* contexts. We could jump across contexts if we could reason about *concatenation*.

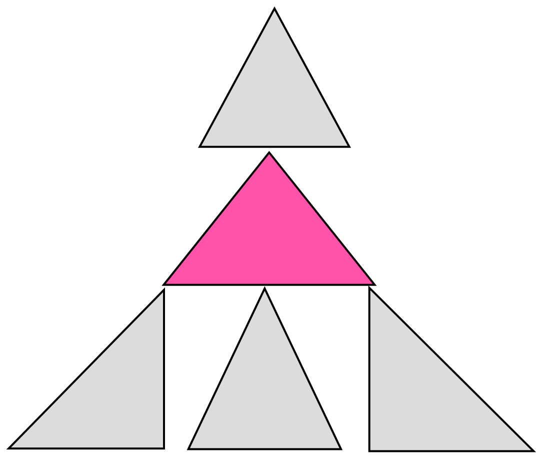
Logic of Subtrees



Mu-calculus formulas can be interpreted at subtrees rather than nodes

Formula → set of subtrees

Modalities argue about full subtrees rooted at children



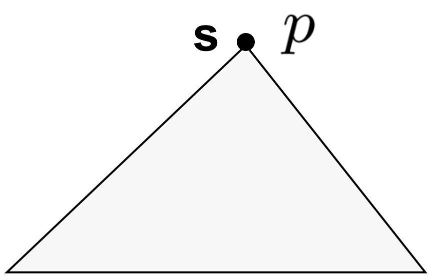
Why not a fixpoint calculus where:

Formulas → sets of summary trees

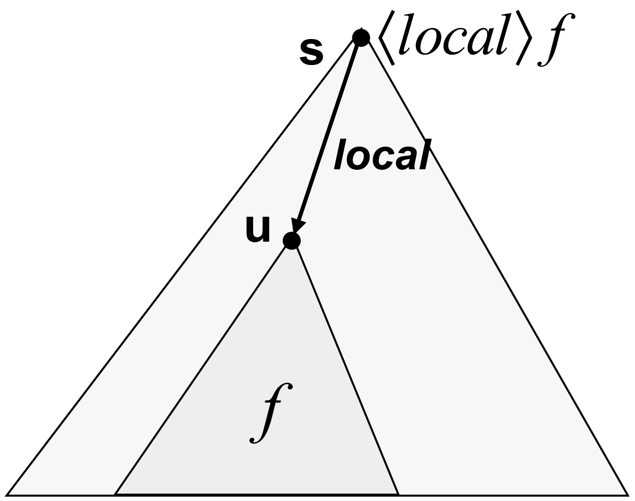
and modalities for concatenation?

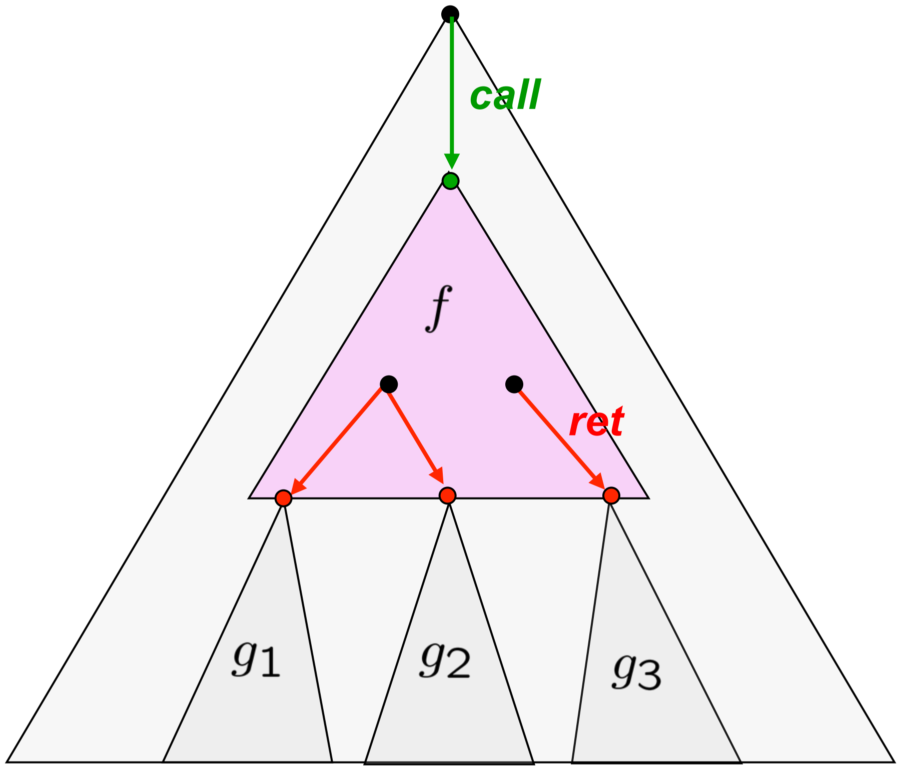
Proposal: NT-mu.

Operations on Summaries



Formulas→ sets of summaries





$$\langle call \rangle f \{g_1, g_2, \ldots\}$$

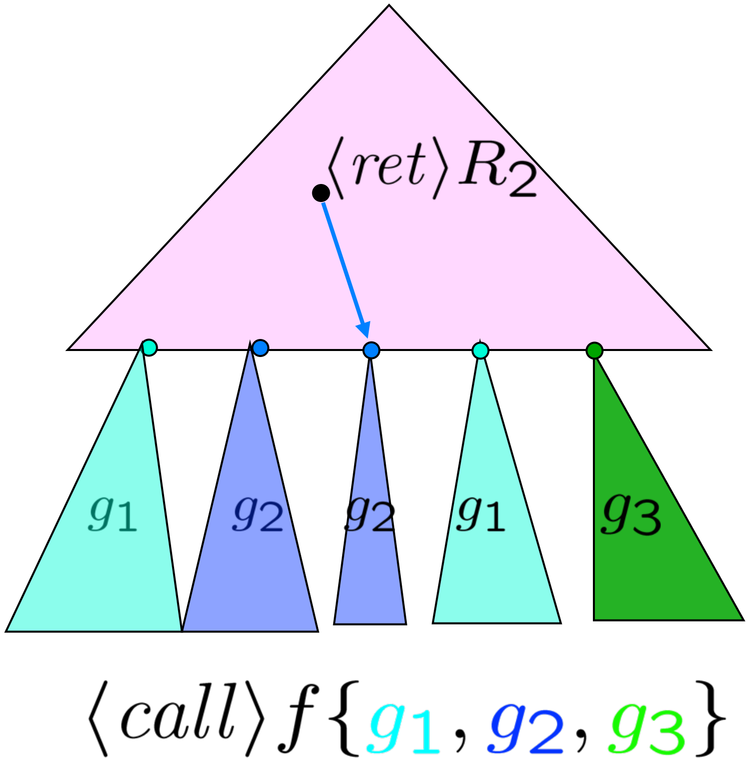
Colored Summary Trees

Number of "leaves" is unbounded

Solution: assign leaves **k** colors

Colors are defined by formulas (obligations upon return)

Within f, we use the propositions R_1, R_2, … R_k to refer to the colors of return leaves


$\langle ret \rangle R_2$
g_1
g_2
g_2
g_1
g_3
$\langle call \rangle f \{g_1, g_2, g_3\}$

Mu-calculus vs NT-mu

$p \qquad \neg p$

$f_1 \wedge f_2 \qquad f_1 \vee f_2$

~~$\langle\rangle f \qquad [] f$~~ $\quad \langle loc \rangle f \qquad [loc] f \qquad \langle ret \rangle R_i \qquad [ret] R_i$

$\langle call \rangle f \{g_1, \ldots, g_k\} \qquad [call] f \{g_1, \ldots, g_k\}$

X

$\mu X . f(X) \qquad \nu X . f(X)$

mu-calculus: fixpoints over full subtrees

NT-mu: fixpoints over summary trees

Semantics of NT-mu

- k-colored summary tree specified by $(s, U_1, \dots U_k)$, where *s* is a tree node, and each U_i is a subset of matching returns of *s*

- Meaning of each formula *f* of NT-mu is a set of summaries
  - $(s, U_1, \dots U_k) \models p$ if label of *s* satisfies *p*
  - Meaning of Boolean connectives is standard
  - $(s, U_1, \dots U_k) \models \langle loc\rangle f$ if *s* has an internal-child *t* s.t. $(t, U_1, \dots U_k) \models f$
  - $(s, U_1, \dots U_k) \models \langle ret\rangle R_i$ if *s* has a return-child *t* s.t. *t* is in U_i
  - $(s, U_1, \dots U_k) \models \langle call\rangle f(g_1, \dots g_m)$ if *s* has a call-child *t* s.t. $(t, V_1 \dots V_m) \models f$ where V_j contains all matching returns *w* of *t* s.t. $(w, U_1, \dots U_k) \models g_j$
  - Formulas define monotonic functions from summary sets to summary sets; fixpoint semantics is standard

- A nested tree T with root r satisfies f if $(r) \models f$

Examples

- There exists a return colored 1: summaries (s,U) s.t. U is non-empty

    f : m X. (<ret> R1 or <loc> X or <call> X {X})

- p is reachable : EF p

    m X. (p or <loc> X or <call> X {} or <call> f {X})

- Local reachability: p is reachable within the same procedural context

    m X. (p or <loc> X or <call> f {X}

Specifying Requirements

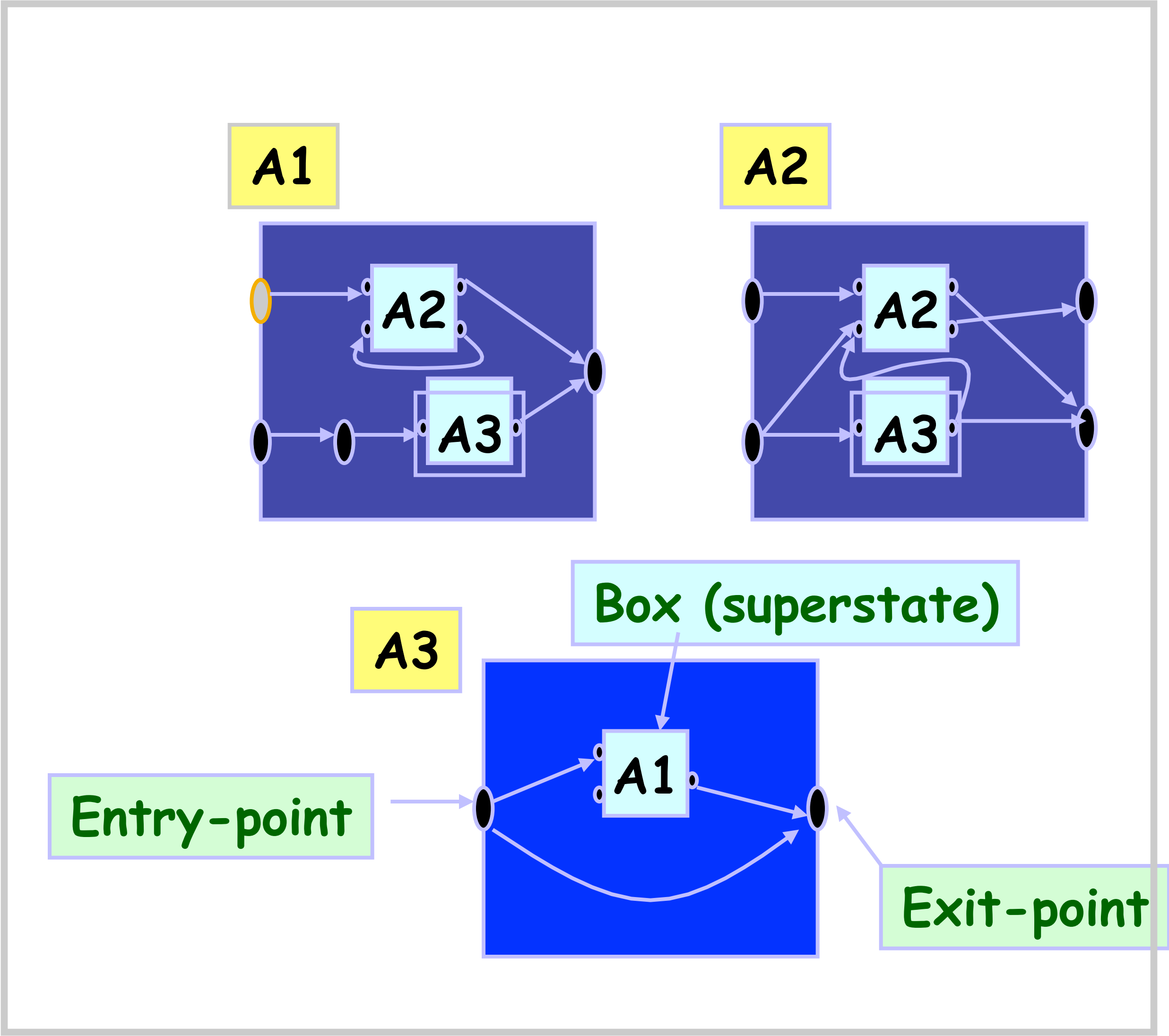
- Branching-time properties that mix local and global paths
- Inter-procedural data-flow analysis
  - Set of program points where expression *e* is very busy (along every path *e* is used before a variable in e gets modified)
  - If *e* contains local variables, this is not definable in mu-calculus
- Stack inspection, access control, stack overflow
- Pre-post conditions (universal as well as branching)

Program Models

Program

```
main() {
  bool y;
  …
  x = P(y);
  …
  z = P(x);
  …
}
bool P(u: bool) {
…
return Q(u);
}
bool Q(w: bool) {
  if …
  else return P(~w)
}
```

Recursive State Machine (RSM)/ Pushdown automaton



Model Checking

- Given an RSM *A* and NT-mu formula *f*, does the nested tree T_A satisfy *f* ?
- Consider a point *a* in a component with exits *u* and *v*
  - *A* sample state of *A* is of the form s.a, where s is a stack of boxes
  - State at any matching return of s.a is either s.u or s.v
- Claim 1: NT-mu is a tree logic, so even though *s.a* may appear at multiple places in T_A, it satisfies the same formulas
- Claim 2: NT-mu formulas are evaluated over summary trees (cannot access nodes beyond matching returns), satisfaction of formula at *s.a* does not depend on the context s

Bisimulation Closure

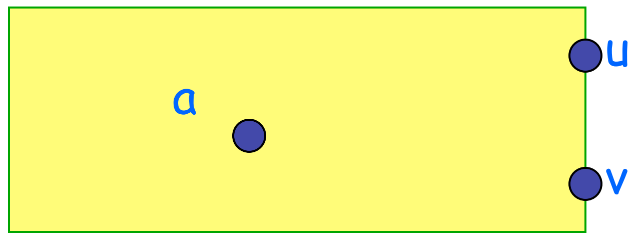
A summary $(s,U_1,\ldots U_k)$ is bismulation-closed if two matching returns w and w' are bisimilar, then w in U_i iff w' in U_i

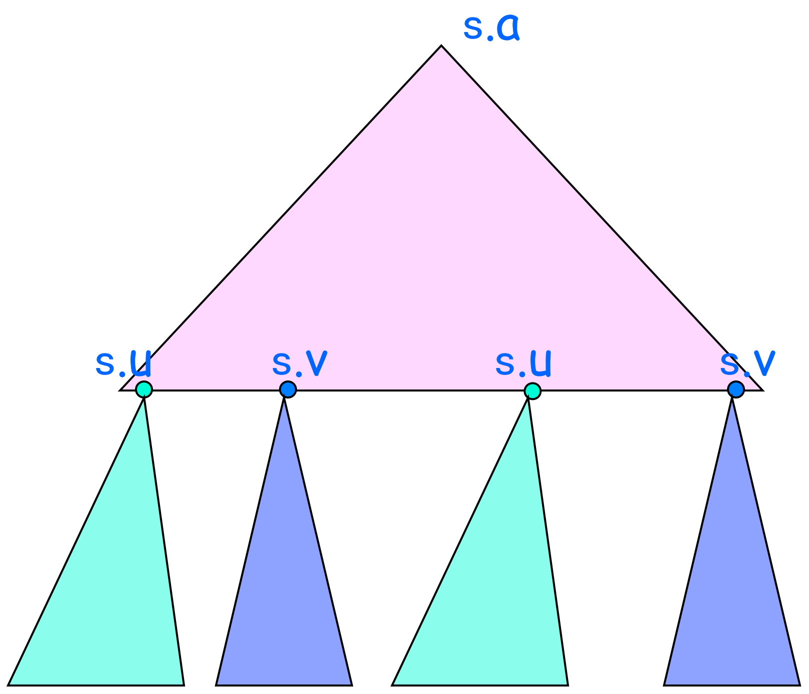
Claim: During fixpoint evaluation, it suffices to consider only bisimulation-closed summaries

Closing each color under bisimulation does not change the truth of formulas

Return nodes corresponding to the same exit are bisimilar

Corollory: Bisimulation-closed summaries have finite representation (colors for each exit)





Model Checking

- Model checking procedure:
  - Consider RSM-summaries of the form *(s,U_1,..U_k),* where *s* is a vertex in a component, and U_i is a subset of exit points
  - Finitely many RSM summaries
  - Evaluate NT-mu formula using standard fixpoint computation
- Model checking RSMs wrt NT-mu is Exptime-complete
  - Same complexity as CTL or mu-calculus model checking
- Recall reachability in NT-mu
  - f: m X. (<ret> R1 or <loc> X or <call> X {X})
  - EF p : m X. (p or <loc> X or <call> X {} or <call> f {X})
  - Local-reach: m X. (p or <loc> X or <call> f {X})
- Evaluation of these over RSM-summaries is the standard way of solving reachability
  - Evaluating f corresponds to pre-computing summaries
  - Global/local reachability are computationally similar

Expressiveness

Thm: NT-mu and APNTA are equally expressive

Corollary: NT-mu can capture every property that the mu-calculus can.

Corollary: CARET (a linear temporal logic of calls and returns, AEM'04) is contained in NT-mu.

Corollary: Satisfiability of NT-mu is undecidable.

NT-mu can express pushdown games

Thm: Expressiveness increases with the number of colors

From NT-mu to APNTA (Proof sketch)

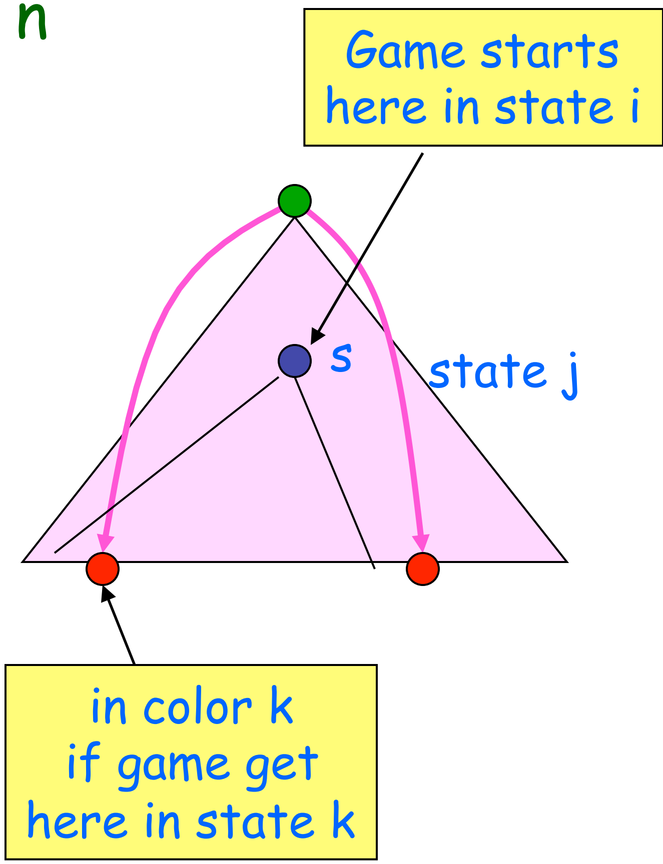
- Given an NT-mu formula f, construct equivalent APNTA A
- States of A are subformulas of f
- Simplify(f,a), where a is an assignment to atomic props
  - Unroll any top-level fixpoint of f
  - Replace each top-level proposition by its T/F value according to a
  - Simplify(f,a) is a positive Boolean comb of terms like <>g and []g
- d_i(f,a) = Simplify(f,a)
- d_c(f{g_1,…g_k},a) = (Simplify(f,a), (g_1,…g_k))

  Evaluate f at call node and send (g_1,..g_k) along nesting edge
- d_r(R_i, (g_1,…g_k),a) = Simplify(g_i,a)

  Retrive i-th return obligation from nesting edge, and evaluate it
- Fixpoints handled using parity condition

From APNTA to NT-mu (Proof sketch)

- Given alternating NTA *A* with Q = {1..n}, accepting by final state, construct a set of least fixpoint equations
- Number of colors (return parameters): n
- For each pair of states, a variable X_{ij}
- Intended meaning: A summary (s, U_1, …U_n) is in X_{ij} iff *A* has a strategy starting at *s* in state *i*, with state *j* along all nested edges to return, to end up in a matching return *s'* in U_k in state *k*
- Write equations among X_{ij} variables so that the lfp captures the intended meaning


Game starts here in state i
s
state j
in color k if game get here in state k

MSO Logic for Nested Trees

Monadic Second Order Logic of Nested Trees

First order variables: x,y,z; Set variables: X,Y,Z...

Atomic formulas: a(x), X(x), x=y, x ->y, x -> y

Logical connectives and quantifiers

Thm: Model-checking even the bisimulation-closed fragment of MSO is undecidable.

Thm: More expressive than NPNTAs.

Thm: Can encode a property not expressible by APNTAs.

Conjecture: Expressiveness of MSO and APNTAs incomparable.

Recap

- Allowing a program to expose call-return summary edges leads to
  - Linear-time: Program is a set of nested words
  - Branching-time: Program is a nested tree
- Nested words arise in other applications: Model for explicit linear and hierarchical orders
- Robust theory of regular languages of nested words
- Powerful fixpoint logic and alternating automata to specify languages of nested trees with decidable model checking problem

Recap

- Papers: Nested words (DLT'06), Nested trees (CAV'06); available from my webpage (caution: definitions/ideas still evolving)

- Interesting offshoot: existing definitions of pushdown tree automata are only "universal" in pushdown component
  - Cannot express "every [is matched by } on some branches and) on some branches"
  - Solution: Branching pushdown tree automata (AC' 06)

- Many, many open/unexplored problems, for example,
  - First-order logics over nested words and nested trees
  - Temporal logics over nested words and nested trees
  - MSO/automata connection for nested trees
  - Edit distances between nested words
  - In which applications can we replace pushdown automata by NWAs
  - Streaming XML, lower bounds on queries…