The Tree Width of Separation Logic with Recursive Definitions

Radu Iosif, Verimag/CNRS, Grenoble

Adam Rogalewicz, Jiri Simacek (Tech. University Brno, CZ)

Why Separation Logic ?

- Industrial-scale program analyzers developped during the last decade: SLAM (Microsoft Research), Astrée (ENS Paris, AIRBUS)

- Either ignore or build very coarse abstractions of the program's heap

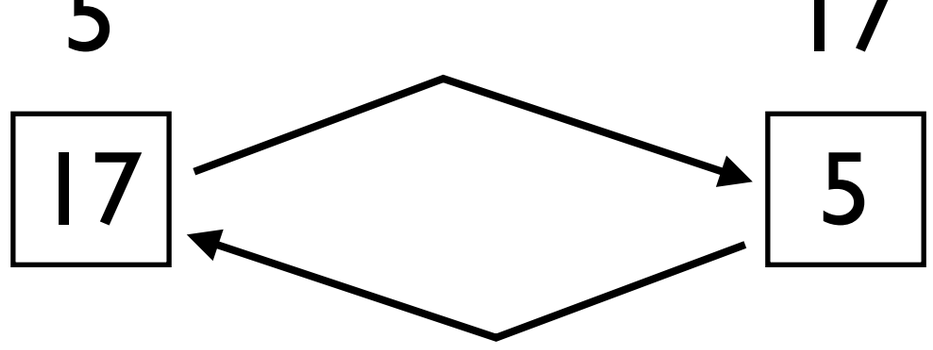
- A lot of software artifacts are beyond the scope of these tools (OS kernels, Web servers, Network apps)

- Reasoning about the heap is hard: compositional methods required to ensure (some degree of) scalability

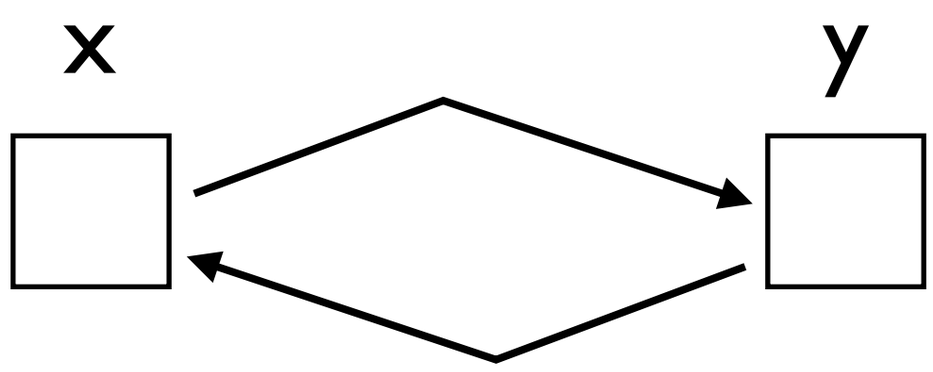
Basic Separation Logic

1 2 3 4 5 ... 17

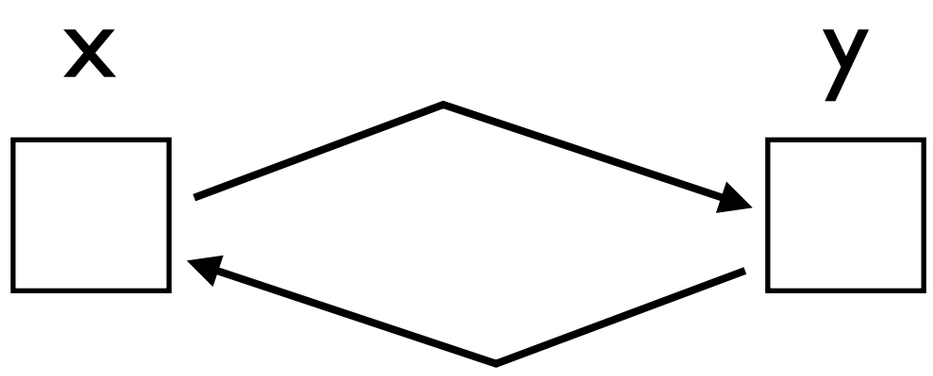
Basic Separation Logic



Basic Separation Logic

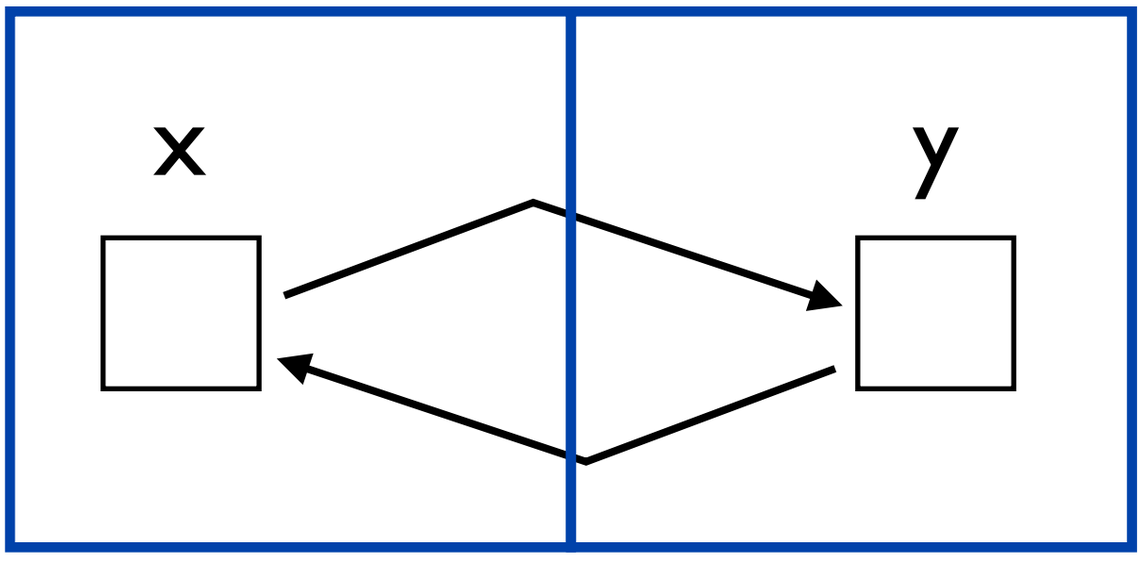


Basic Separation Logic



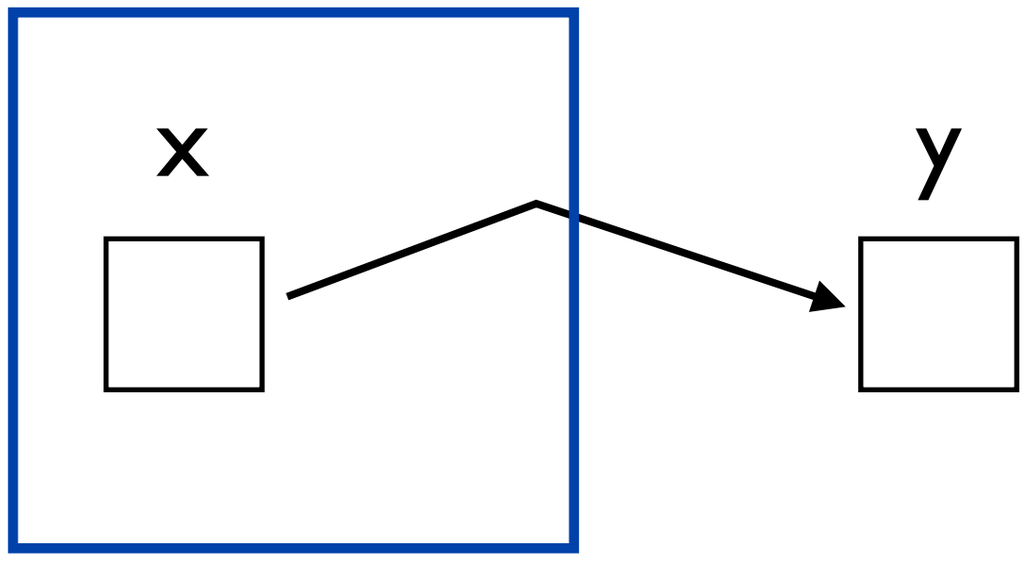
$x \rightarrow y * y \rightarrow x$

Basic Separation Logic



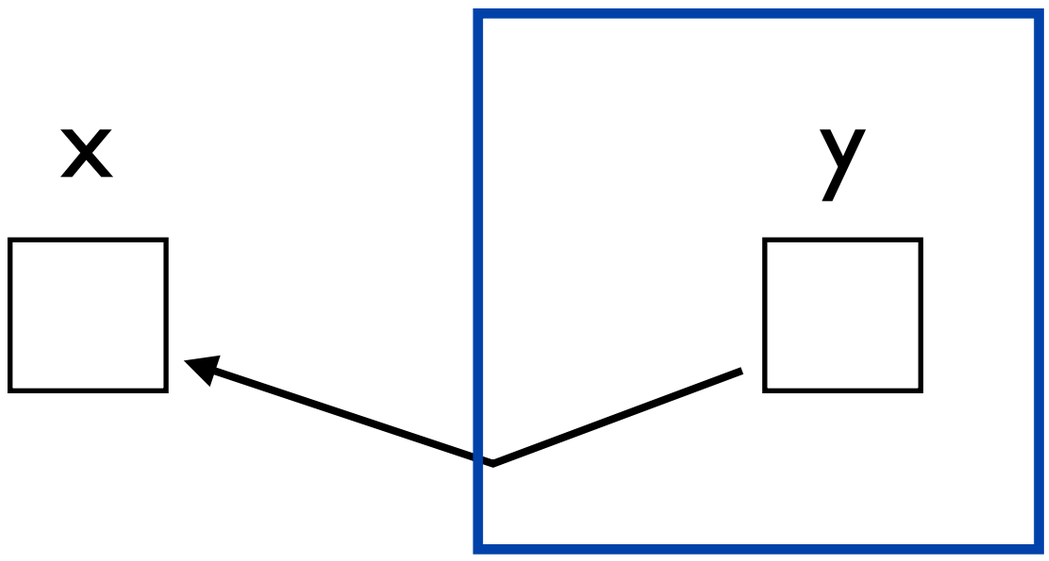
$x \rightarrow y * y \rightarrow x$

Basic Separation Logic



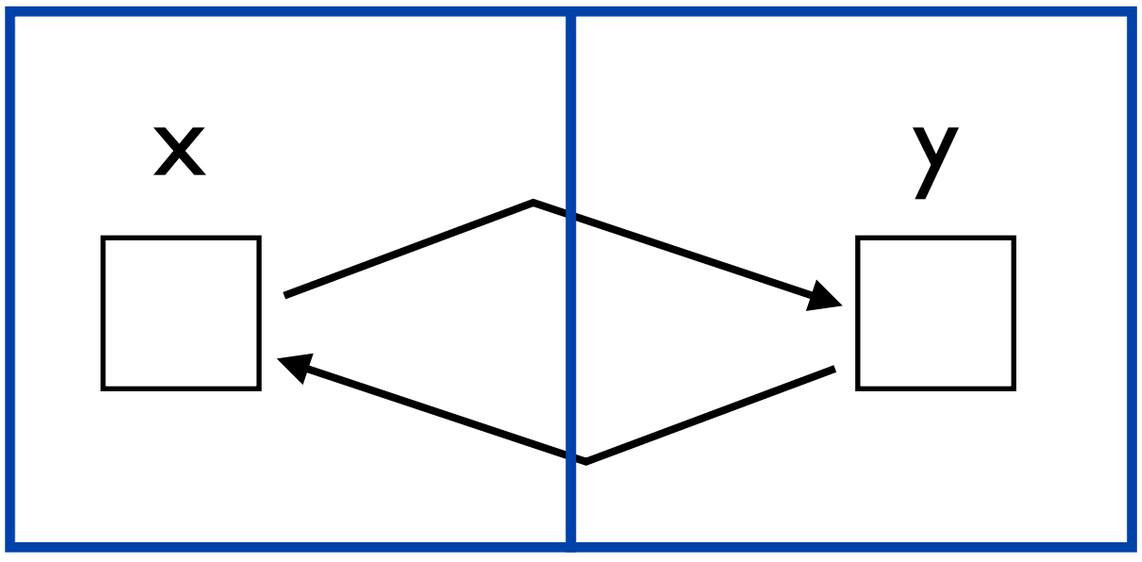
$x \rightarrow y$

Basic Separation Logic



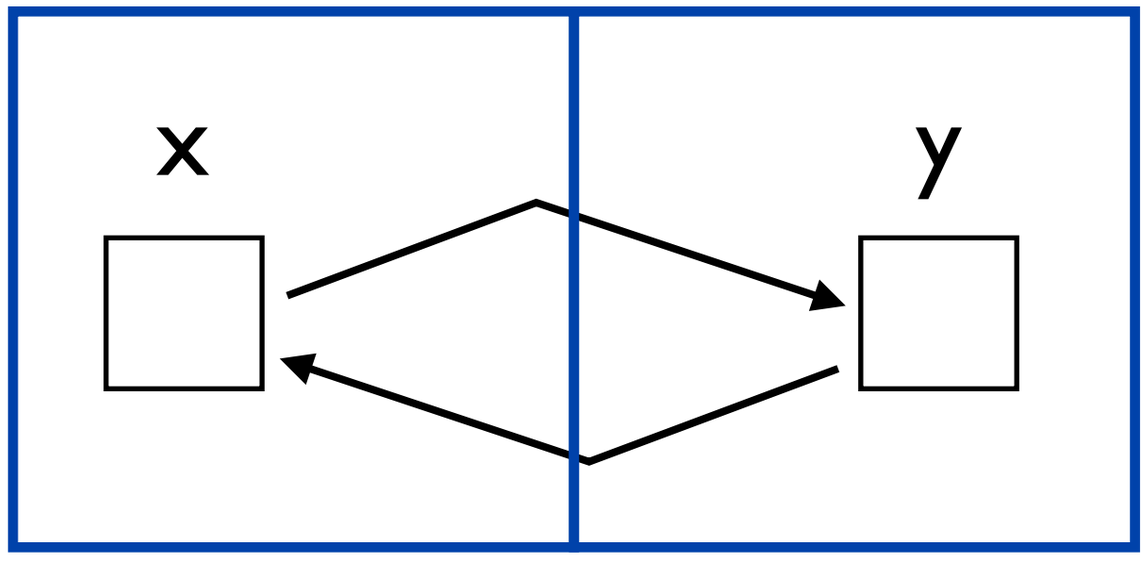
$y \rightarrow x$

Basic Separation Logic



$x \rightarrow y * y \rightarrow x$

Basic Separation Logic



$x \rightarrow y * y \rightarrow x \implies x \neq y$

Basic Separation Logic

Adds the following primitives to first-order logic:

- emp = the heap is empty
- $x \rightarrow y$ = the heap has exactly one cell x, pointing to y
- $\phi * \psi$ = the heap can be split such that ϕ is true on one part, and ψ is true on the other part

Basic Separation Logic

$$\varphi \equiv \exists \mathbf{x,y,z} \,.\, \bigvee_{i>0} (\Sigma_i \wedge \Pi_i)$$

$$\Sigma \equiv x_1^{\,l} \rightarrow (y_1^{\,l},\ldots,y_n^{\,l}) * \ldots * x_1^{\,k} \rightarrow (y_1^{\,k},\ldots,y_n^{\,k}) \qquad \text{(spatial)}$$

$$\Pi \equiv x_1 = y_1 \wedge \ldots \wedge x_n = y_n \wedge z_1 \neq t_1 \wedge \ldots \wedge z_m \neq t_m \qquad \text{(pure)}$$

Describe only "heaplets" bounded by the size of φ

Adding Recursive Definitions

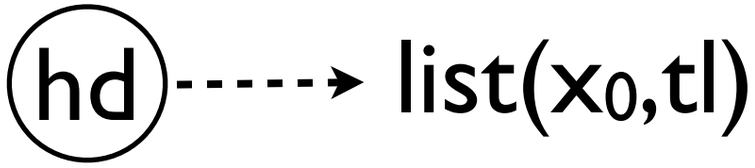
list(hd,tl) ::= emp /\ hd=tl (singly-linked list)

| ∃x . hd→x * list(x,tl)

Adding Recursive Definitions

list(hd,tl) ::= emp /\ hd=tl (singly-linked list)

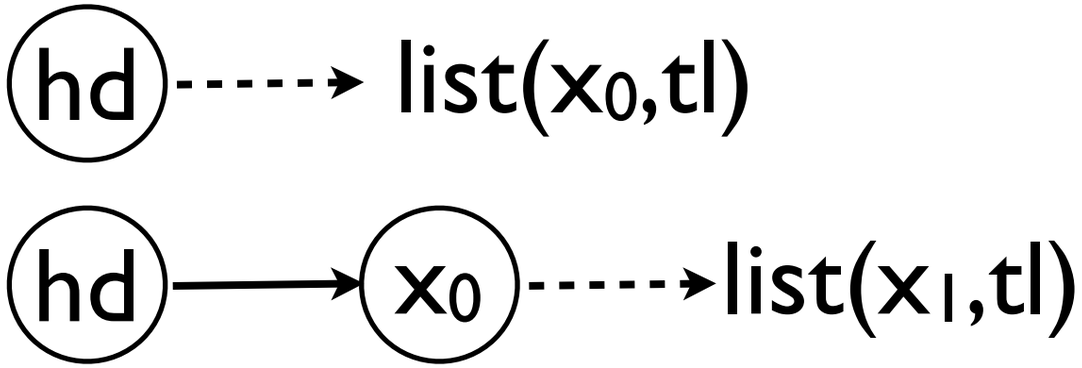
| ∃x . hd→x * list(x,tl)



Adding Recursive Definitions

list(hd,tl) ::= emp /\ hd=tl (singly-linked list)

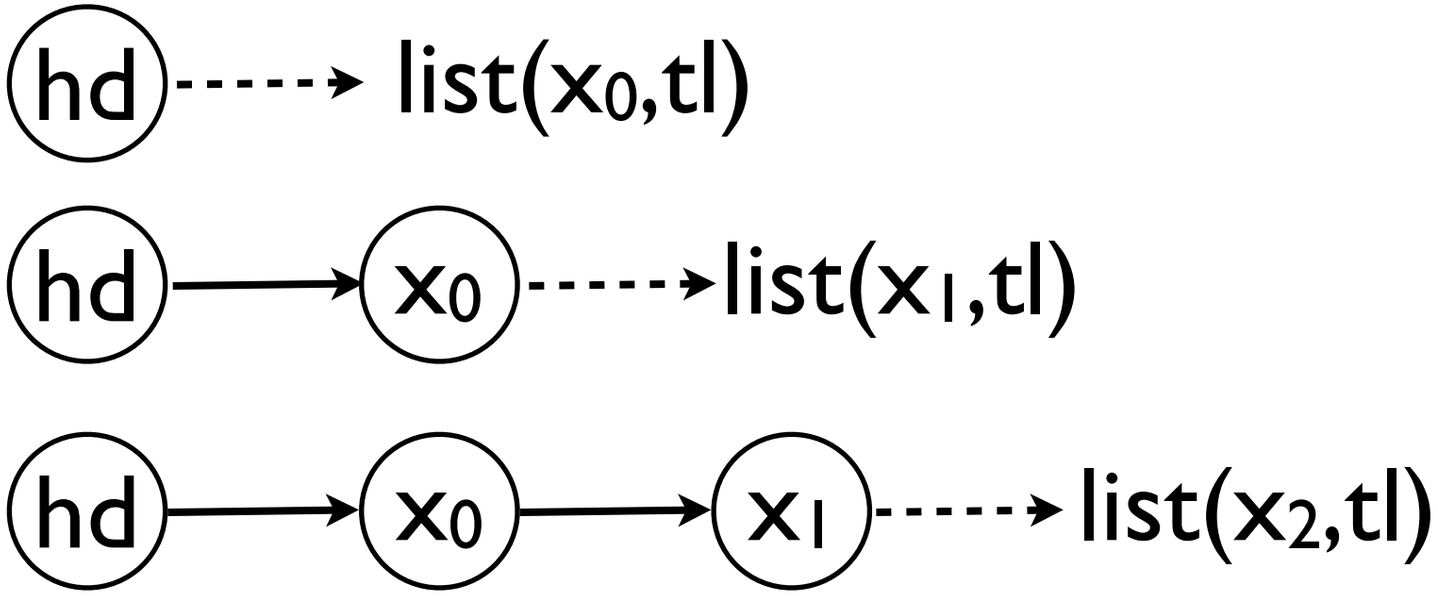
| ∃x . hd→x * list(x,tl)



Adding Recursive Definitions

list(hd,tl) ::= emp /\ hd=tl (singly-linked list)

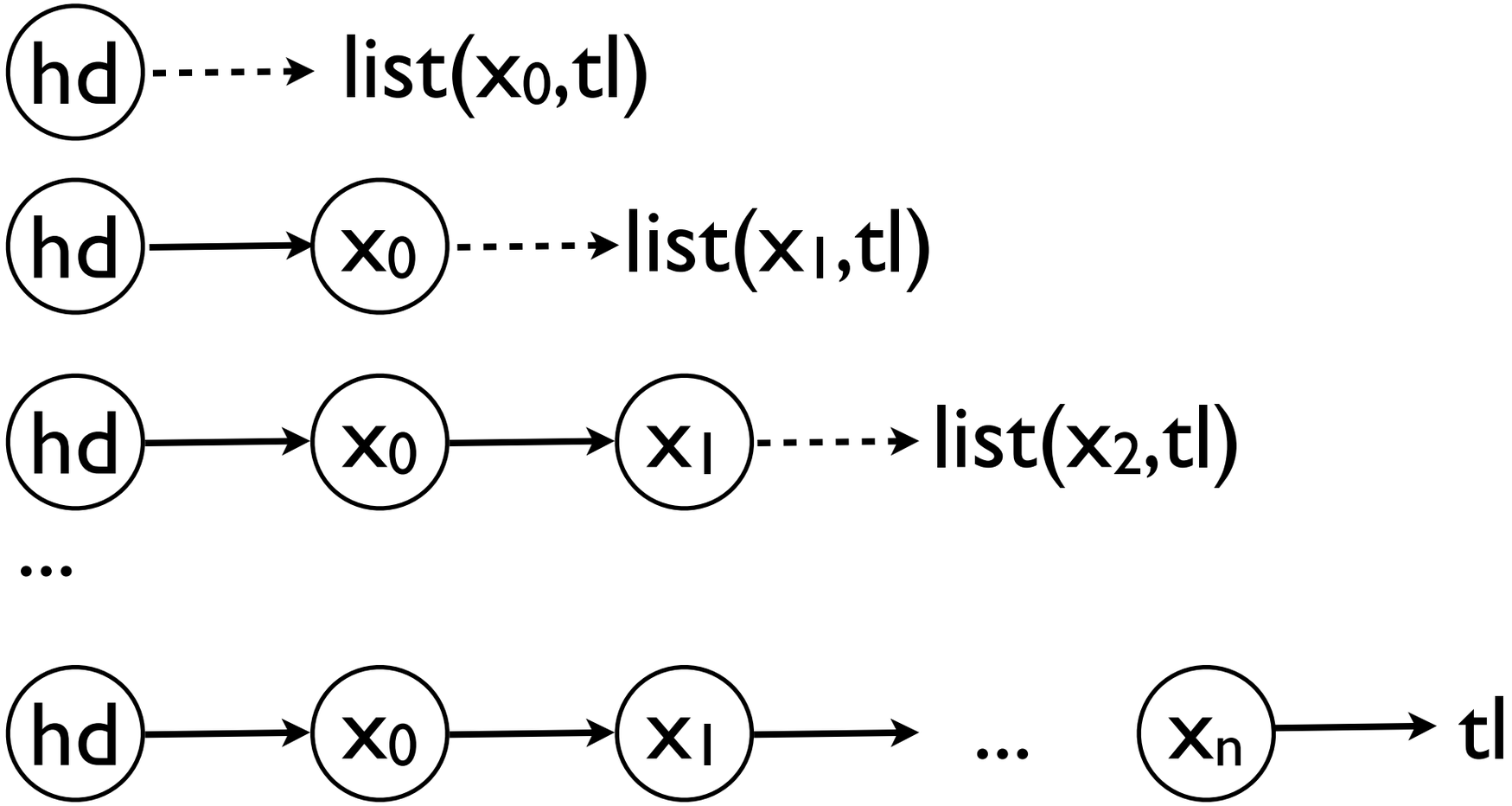
| ∃x . hd→x * list(x,tl)



Adding Recursive Definitions

list(hd,tl) ::= emp /\ hd=tl (singly-linked list)

| ∃x . hd→x * list(x,tl)



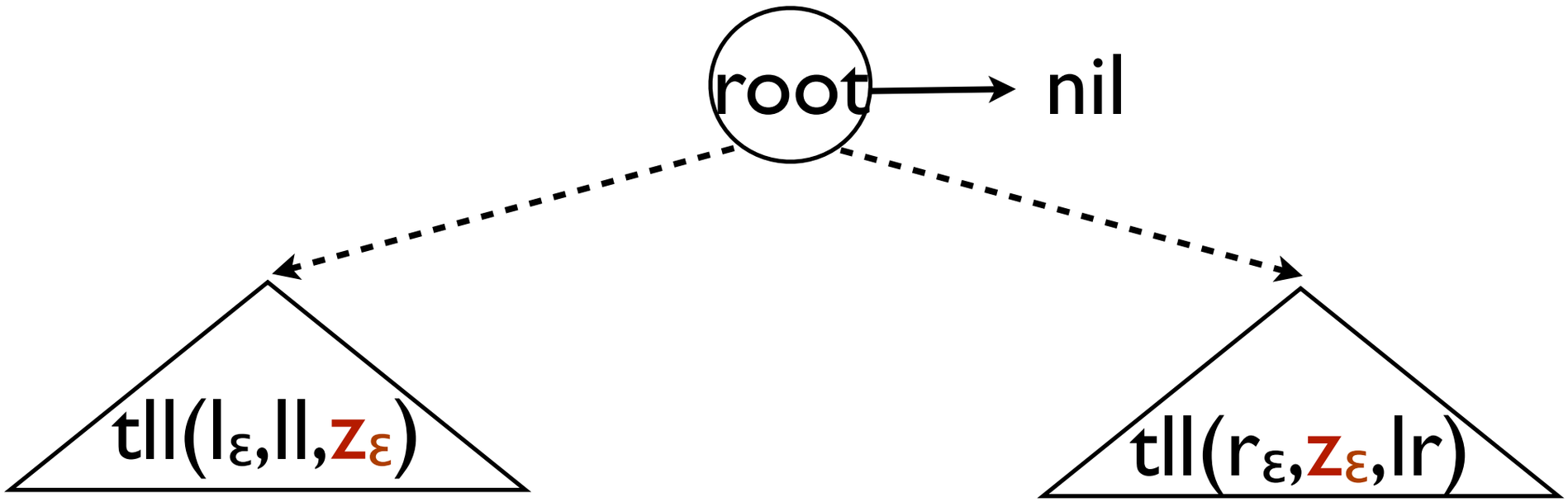
Adding Recursive Definitions

$$\mathrm{tll}(\mathrm{root},\mathrm{ll},\mathrm{lr}) ::= \mathrm{root}\rightarrow(\mathrm{nil},\mathrm{nil},\mathrm{lr}) \wedge \mathrm{root}=\mathrm{ll}$$
$$\mid \exists \mathrm{l},\mathrm{r},\mathrm{z} \,.\, \mathrm{root}\rightarrow(\mathrm{l},\mathrm{r},\mathrm{nil}) * \mathrm{tll}(\mathrm{l},\mathrm{ll},\mathrm{z}) * \mathrm{tll}(\mathrm{r},\mathrm{z},\mathrm{lr})$$

Adding Recursive Definitions

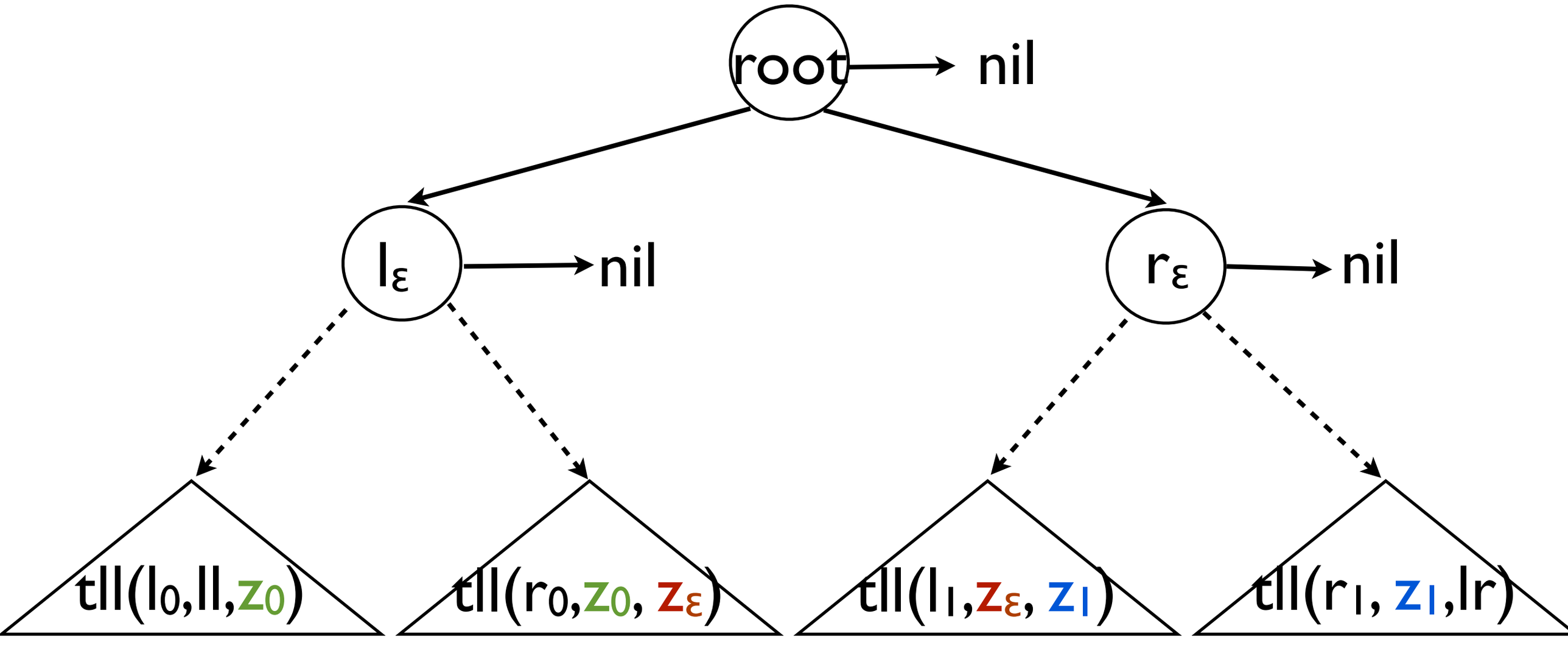
tll(root,ll,lr) ::= root→(nil,nil,lr) /\ root=ll

|∃l,r,z . root→(l,r,nil) * tll(l,ll,z) * tll(r,z,lr)



Adding Recursive Definitions

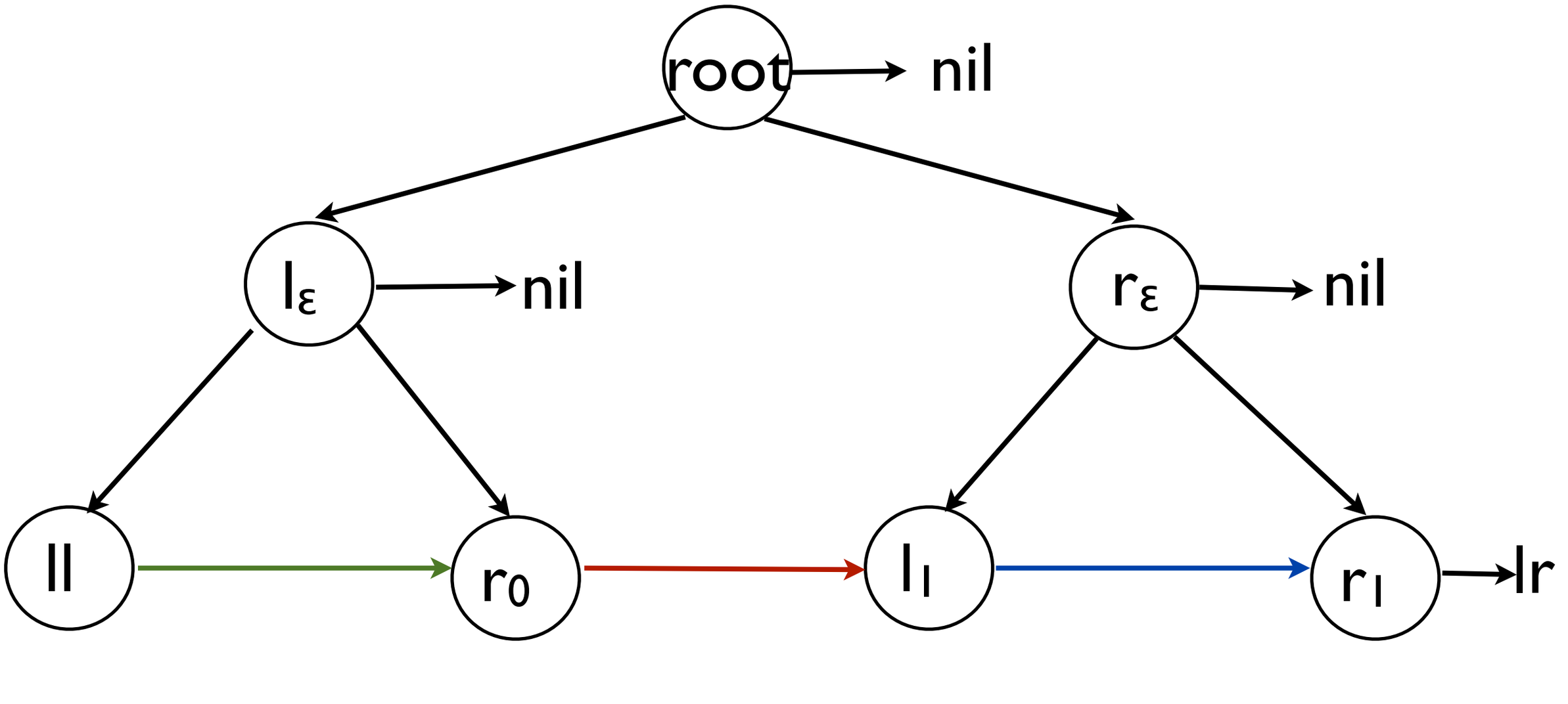
$$\mathrm{tll}(root,ll,lr) ::= root \rightarrow (nil,nil,lr) \wedge root=ll$$
$$\quad | \exists l,r,z \,.\, root \rightarrow (l,r,nil) * \mathrm{tll}(l,ll,z) * \mathrm{tll}(r,z,lr)$$



Adding Recursive Definitions

tll(root,ll,lr) ::= root→(nil,nil,lr) /\ root=ll

|∃l,r,z . root→(l,r,nil) * tll(l,ll,z) * tll(r,z,lr)



Systems of Recursive Definitions

$$P_1(x_{1,1}, \ldots, x_{1,n1}) ::= |_{j=1}^{m1} R_{1,j}(x_{1,1}, \ldots, x_{1,n1})$$

$$\ldots$$

$$P_k(x_{k,1}, \ldots, x_{k,nk}) ::= |_{j=1}^{mk} R_{k,j}(x_{k,1}, \ldots, x_{k,nk})$$

$$R_{i,j}(x_{i,1} \ldots x_{i,ni}) \equiv \exists \mathbf{z} \,.\, \varphi(\mathbf{x},\mathbf{z}) * P_{i,1}(\mathbf{x},\mathbf{z}) * \ldots * P_{i,m}(\mathbf{x},\mathbf{z})$$

Systems of Recursive Definitions

$$P_1(x_{1,1}, \ldots, x_{1,n1}) ::= |_{j=1}^{m1} R_{1,j}(x_{1,1}, \ldots, x_{1,n1})$$

$$\ldots$$

$$P_k(x_{k,1}, \ldots, x_{k,nk}) ::= |_{j=1}^{mk} R_{k,j}(x_{k,1}, \ldots, x_{k,nk})$$

$$R_{i,j}(x_{i,1} \ldots x_{i,ni}) \equiv \exists \mathbf{z} \,.\, \varphi(\mathbf{x},\mathbf{z}) * P_{i,1}(\mathbf{x},\mathbf{z}) * \ldots * P_{i,m}(\mathbf{x},\mathbf{z})$$

basic SL

recursive predicates

Systems of Recursive Definitions

$$P_1(x_{1,1}, \ldots, x_{1,n1}) ::= |_{j=1}^{m1} R_{1,j}(x_{1,1}, \ldots, x_{1,n1})$$

$$\ldots$$

$$P_k(x_{k,1}, \ldots, x_{k,nk}) ::= |_{j=1}^{mk} R_{k,j}(x_{k,1}, \ldots, x_{k,nk})$$

$$R_{i,j}(x_{i,1} \ldots x_{i,ni}) \equiv \exists \mathbf{z} \,.\, \varphi(\mathbf{x},\mathbf{z}) * P_{i,1}(\mathbf{x},\mathbf{z}) * \ldots * P_{i,m}(\mathbf{x},\mathbf{z})$$

Systems of Recursive Definitions

$$P_1(x_{1,1}, \ldots, x_{1,n1}) ::= |_{j=1}^{m1} R_{1,j}(x_{1,1}, \ldots, x_{1,n1})$$

$$\ldots$$

$$P_k(x_{k,1}, \ldots, x_{k,nk}) ::= |_{j=1}^{mk} R_{k,j}(x_{k,1}, \ldots, x_{k,nk})$$

$$R_{i,j}(x_{i,1} \ldots x_{i,ni}) \equiv \exists \mathbf{z} \,.\, \varphi(\mathbf{x},\mathbf{z}) * P_{i,1}(\mathbf{x},\mathbf{z}) * \ldots * P_{i,m}(\mathbf{x},\mathbf{z})$$

The size of the system:

$$||\{P_1, \ldots, P_k\}|| = \max \{ ||R_{i,j}|| \mid 1 \leq i \leq k,\ 1 \leq j \leq n_i \}$$

where $||R_{i,j}|| = ||\mathbf{x}|| + ||\mathbf{z}||$

SL with Recursive Definitions

$$\exists x_0 \ldots x_n \,.\, \forall_{i>0}\, \varphi_i(x_0 \ldots x_n) * P_{i,1}(x_0 \ldots x_n) * \ldots * P_{i,k}(x_0 \ldots x_n)$$

SL with Recursive Definitions

$$\exists x_0 \ldots x_n \,.\, \bigvee_{i>0} \varphi_i(x_0 \ldots x_n) * P_{i,1}(x_0 \ldots x_n) * \ldots * P_{i,k}(x_0 \ldots x_n)$$

basic SL

recursive predicates

SL with Recursive Definitions

$$\exists x_0 \ldots x_n \,.\, \bigvee_{i>0} \varphi_i(x_0 \ldots x_n) * P_{i,1}(x_0 \ldots x_n) * \ldots * P_{i,k}(x_0 \ldots x_n)$$

basic SL

recursive predicates

- A natural way of specifying recursive data structures (lists, trees, and beyond)
- Unfolding the definitions => tree structure
- Non-local edges occur due to parameter sharing
- Possibility of describing general graph-like structures

Program Verification with SL

i = CreateTree();

j = CreateTree();

r = NewCell(i,j);

Program Verification with SL

{emp} x = CreateTree() {tree(x)}

{emp}
i = CreateTree();

j = CreateTree();

r = NewCell(i,j);

{tree(r)}

Program Verification with SL

{emp} x = CreateTree() {tree(x)}

{emp}
i = CreateTree();
{tree(i)}

j = CreateTree();

r = NewCell(i,j);

Program Verification with SL

{emp} x = CreateTree() {tree(x)}

{emp}
i = CreateTree();
{tree(i)}
{tree(i) * emp} (new frame)
j = CreateTree();

r = NewCell(i,j);

Program Verification with SL

{emp} x = CreateTree() {tree(x)}

{emp}
i = CreateTree();
{tree(i)}
{tree(i) * emp} (new frame)
j = CreateTree();
{tree(i) * tree(j)}

r = NewCell(i,j);

Program Verification with SL

{emp} x =NewCell(y,z) {x→(y,z)}

{emp}
i = CreateTree();
{tree(i)}
{tree(i) * emp} (new frame)
j = CreateTree();
{tree(i) * tree(j)}
{emp * tree(i) * tree(j)} (new frame)
r = NewCell(i,j);

Program Verification with SL

{emp} x =NewCell(y,z) {x→(y,z)}

{emp}
i = CreateTree();
{tree(i)}
{tree(i) * emp} (new frame)
j = CreateTree();
{tree(i) * tree(j)}
{emp * tree(i) * tree(j)} (new frame)
r = NewCell(i,j);
{r→(i,j) * tree(i) * tree(j)}

Program Verification with SL

{emp} x =NewCell(y,z) {x→(y,z)}

{emp}
i = CreateTree();
{tree(i)}
{tree(i) * emp} (new frame)
j = CreateTree();
{tree(i) * tree(j)}
{emp * tree(i) * tree(j)} (new frame)
r = NewCell(i,j);
{r→(i,j) * tree(i) * tree(j)}
{tree(r)} (entailment)

Decidability of SLRD

- Satisfiability of an existential formula
- Validity of entailments between existential formulae
- Proof ingredients:

1. for every existential SLRD formula φ and every model $M \models_{sl} \varphi$, we have $tw(M) \leq f(||\varphi||)$, for some computable function f (with some restrictions)

2. for every existential SLRD formula φ there exists an MSO formula (on graphs) ψ such that:

$$M \models_{sl} \varphi \text{ iff } M \models_{mso} \psi$$

Tree Decomposition of a Graph

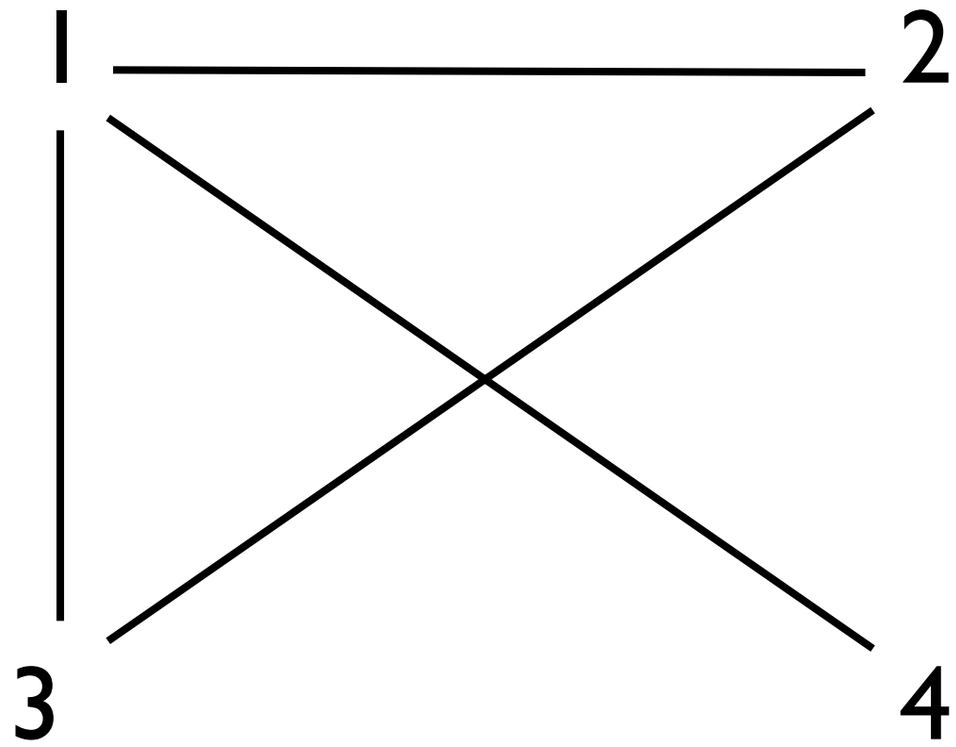
Given a graph $G = (V,E)$, a tree $t : N^* \rightarrow 2^V$ such that:

1. $V = U_{p \in dom(t)} t(p)$
2. for each $(u,v) \in E$ exists $p \in dom(t)$. $u,v \in t(p)$
3. for each $p,q,r \in dom(t)$, q is on the path from p to r, $t(p) \cap t(r) \subseteq t(q)$

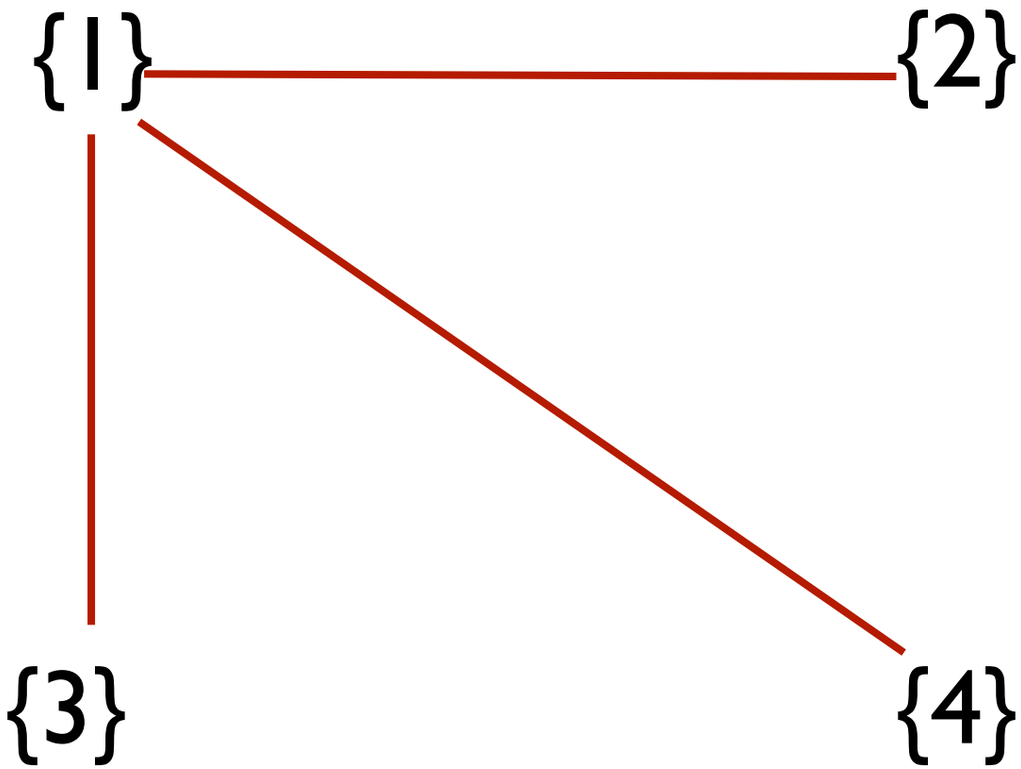
The width of the decomposition t is $max_{p \in dom(t)} \{||t(p)|| - 1\}$

The tree width of G is $min\{width(t) \mid t \text{ is a tree dec. of } G\}$

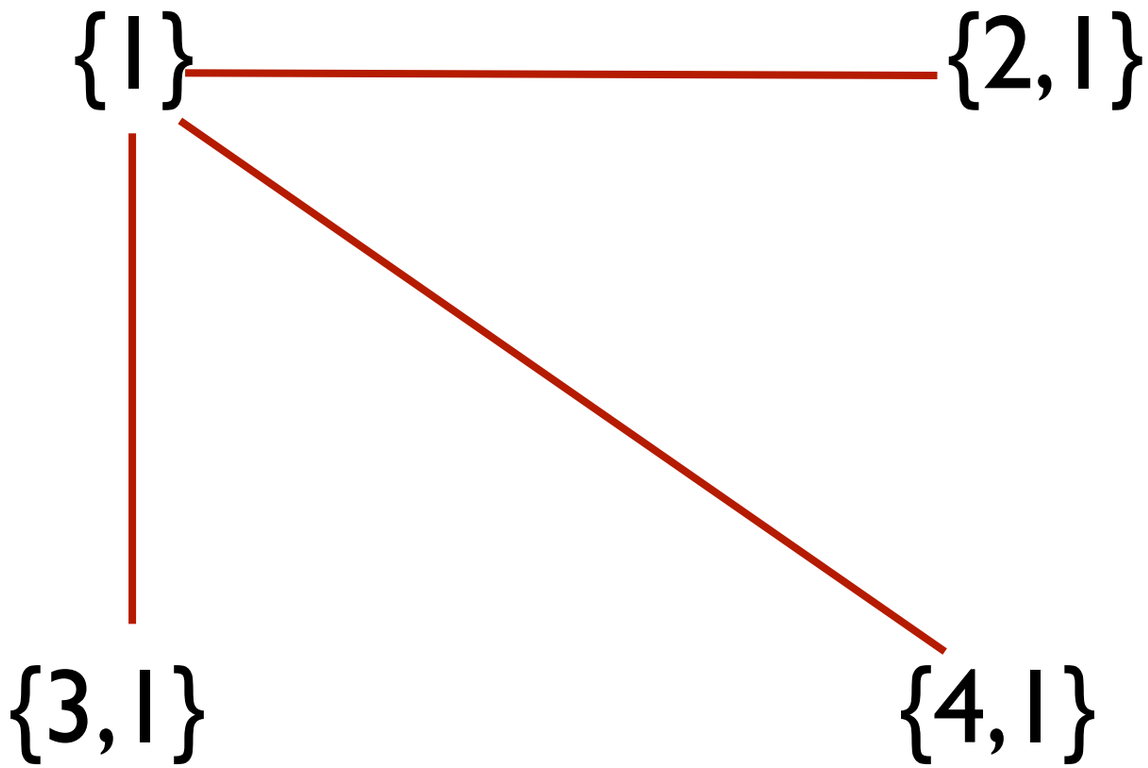
Tree Decomposition of a Graph



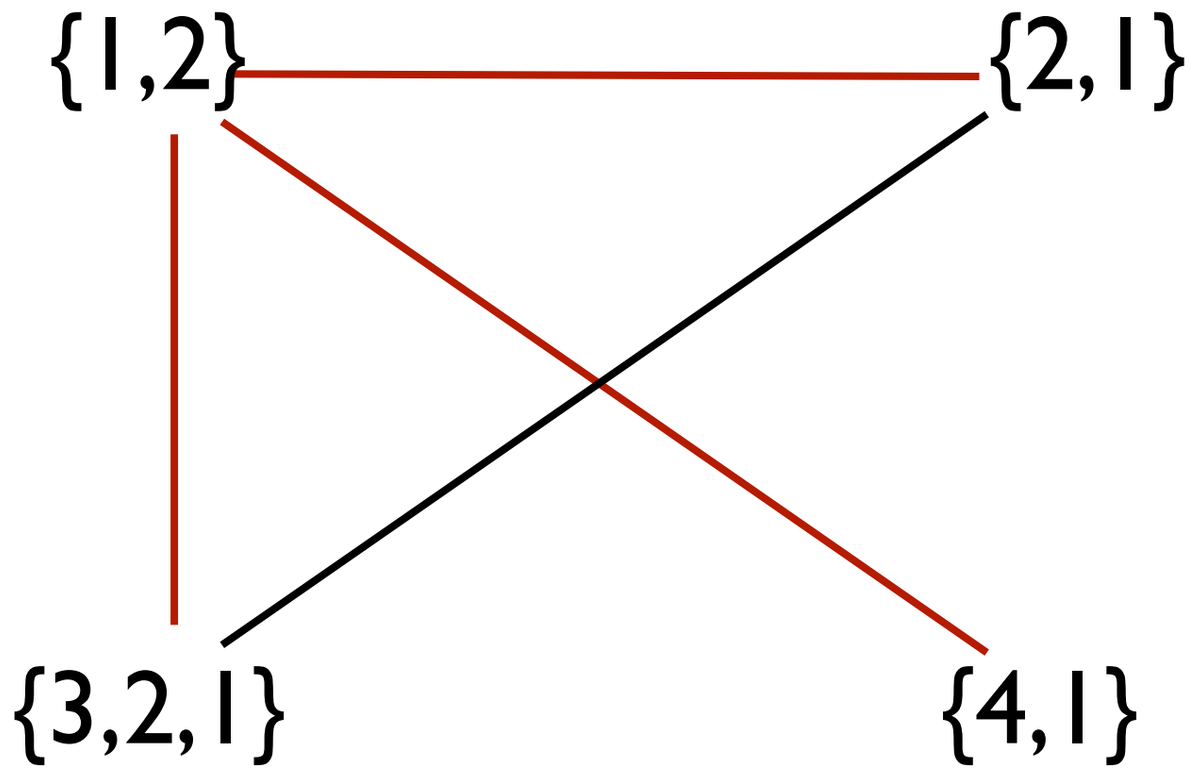
Tree Decomposition of a Graph



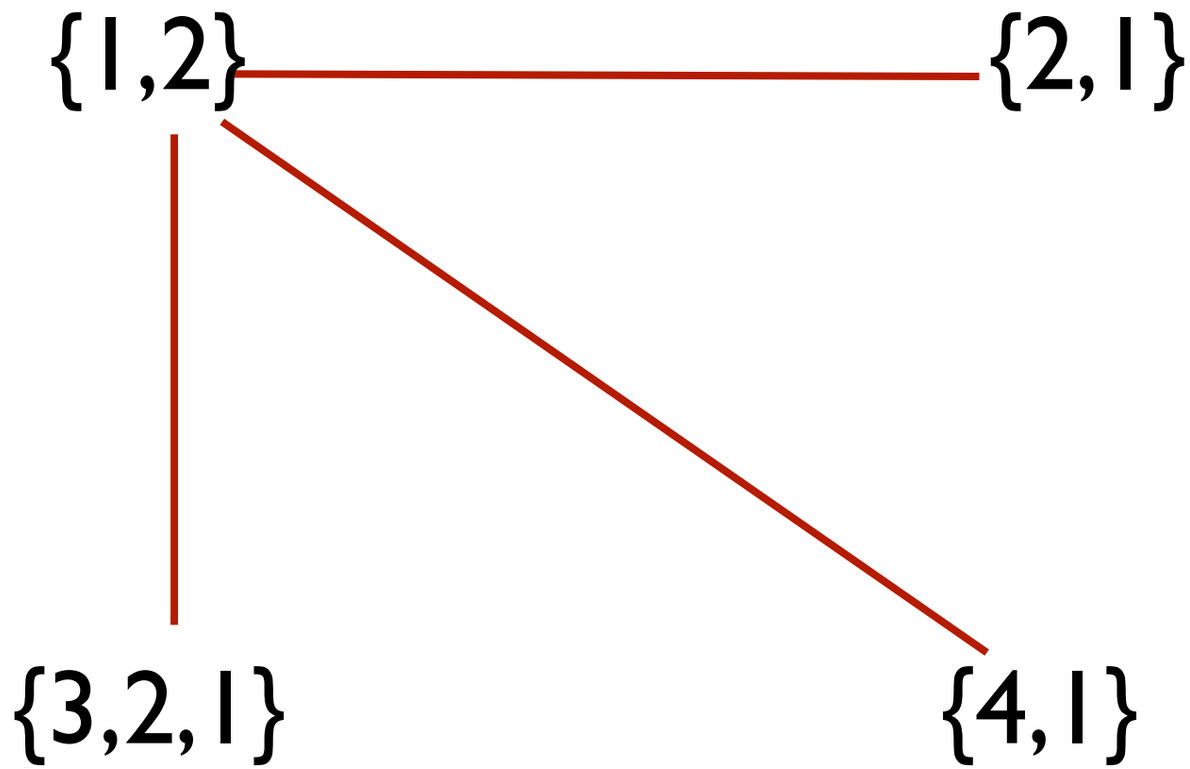
Tree Decomposition of a Graph



Tree Decomposition of a Graph



Tree Decomposition of a Graph

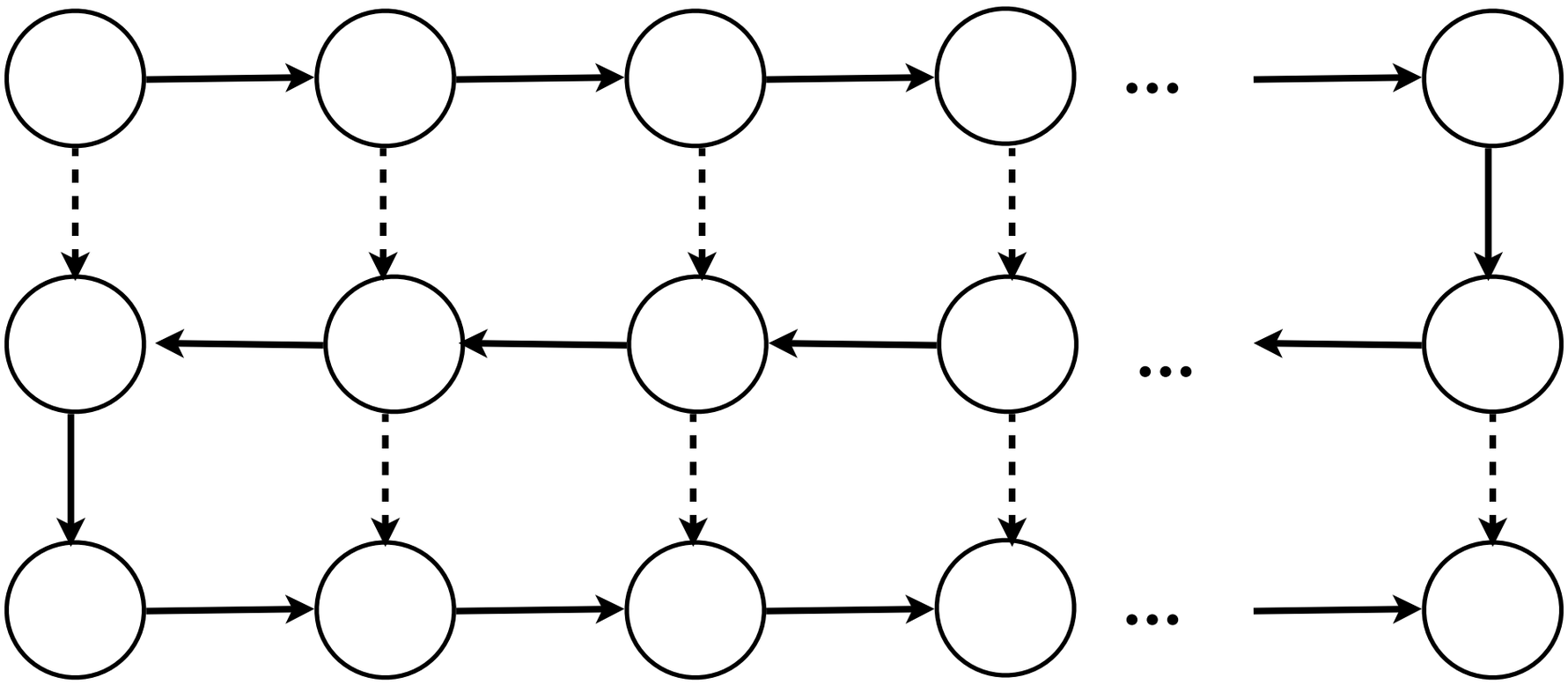


MSO and Bounded Tree Width

- A class of graphs C has bounded tree width iff there exists a constant $k > 0$ such that for all $G \in C$, $tw(G) \leq k$
- Given an MSO formula φ, it is decidable whether φ has a model of tree width less than or equal to k (Courcelle's Theorem)

SLRD with Unbounded TW

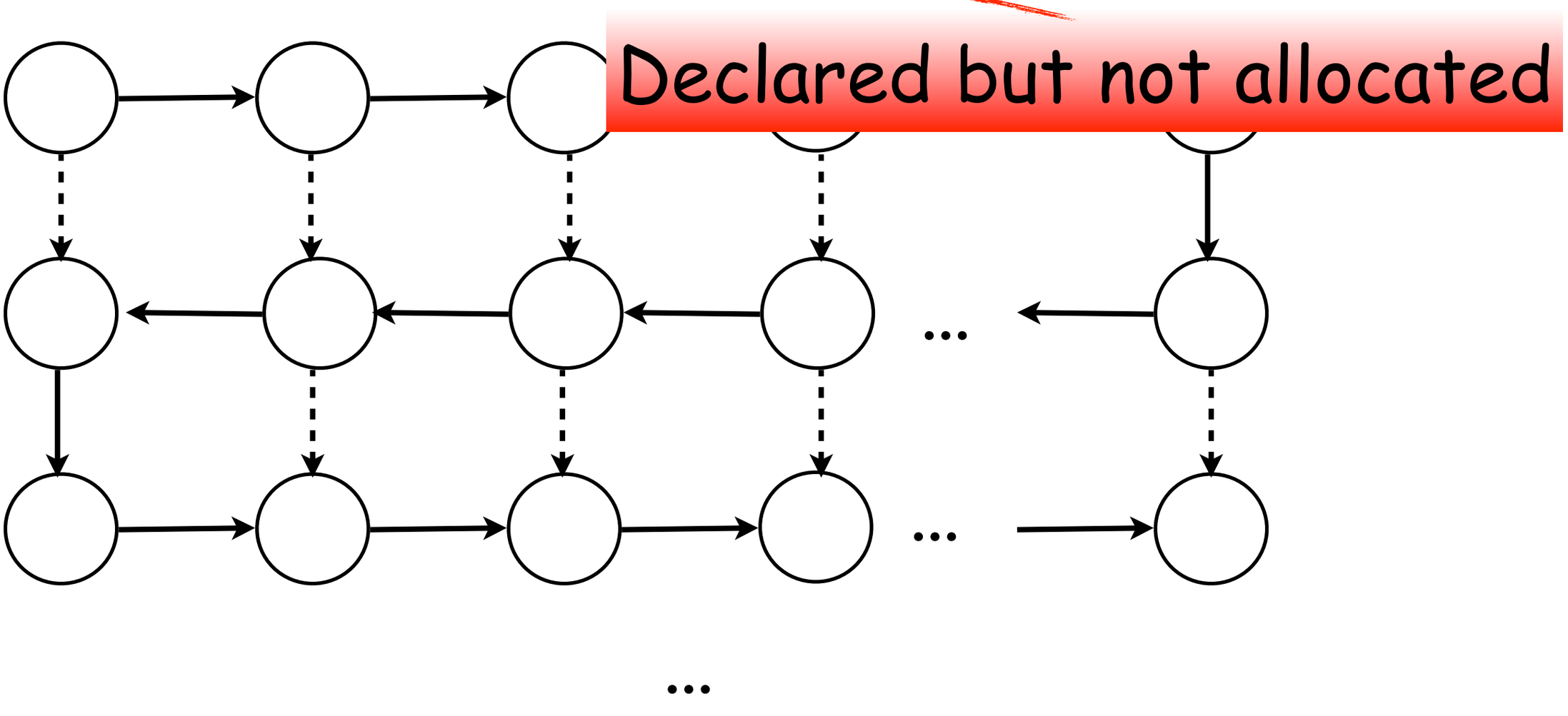
lle(x) ::= x→(nil,nil) | ∃y,z . x→(y,z) * lle(y)



NxN squared grids do not have bounded tree width

SLRD with Unbounded TW

$$lle(x) ::= x \rightarrow (nil,nil) \mid \exists y,z \,.\, x \rightarrow (y,z) * lle(y)$$



NxN squared grids do not have bounded tree width

SLRD with Bounded TW

1. Well-establishment: all existentially quantified variables in some recursive rule are eventually allocated

   - static check sufficient to ensure this condition

2. Progress: each rule allocates exactly one cell

3. Connectedness: there is at least one edge in the graph towards each recursive call, e.g.:

SLRD with Bounded TW

1. Well-establishment: all existentially quantified variables in some recursive rule are eventually allocated

   - static check sufficient to ensure this condition

2. Progress: each rule allocates exactly one cell

3. Connectedness: there is at least one edge in the graph towards each recursive call, e.g.:

$$\text{bad\_list}(x) ::= x \rightarrow \text{nil} \mid \exists y \,.\, x \rightarrow \text{nil} * \text{bad\_list}(y)$$

SLRD with Bounded TW

1. Well-establishment: all existentially quantified variables in some recursive rule are eventually allocated

   - static check sufficient to ensure this condition

2. Progress: each rule allocates exactly one cell

3. Connectedness: there is at least one edge in the graph towards each recursive call, e.g.:

~~bad_list(x) ::= x→nil | ∃y . x→nil * bad_list(y)~~

good_list(x) ::= x→nil | ∃y . x→y * good_list(y)

Decidability of Entailments

$\varphi \models_{sl} \Phi$ is valid?

Decidability of Entailments

$\varphi \models_{sl} \Phi$ is valid?

$\varphi_{mso} \wedge \neg\Phi_{mso}$ is not satisfiable

Decidability of Entailments

$\varphi \models_{sl} \Phi$ is valid?

$\varphi_{mso} \wedge \neg\Phi_{mso}$ is not satisfiable

- $M \models_{sl} \varphi$ iff $M \models_{mso} \varphi_{mso}$
- $tw(M) \leq f(||\varphi||)$
- Satisfiability of $\varphi_{mso} \wedge \neg\Phi_{mso}$ is decidable (Courcelle Thm)

Bounding the Tree Width

$tll(root,ll,lr) ::= \exists l,r,z\ .\ root \rightarrow (l,r,nil) * tll(l,ll,z) * tll(r,z,lr) \mid root \rightarrow (nil,nil,lr) \wedge root = ll$

$\exists l_\varepsilon, r_\varepsilon, z_\varepsilon\ .\ root \rightarrow (l_\varepsilon, r_\varepsilon, nil) \wedge$
$ll_\varepsilon = ll_0 \wedge z_\varepsilon = lr_0 \wedge z_\varepsilon = ll_1 \wedge lr_\varepsilon = lr_1$

$\exists l_0, r_0, z_0\ .\ l_\varepsilon \rightarrow (l_0, r_0, nil) \wedge$
$ll_0 = ll_{00} \wedge z_0 = lr_{00} \wedge z_0 = ll_{01} \wedge lr_0 = lr_{01}$

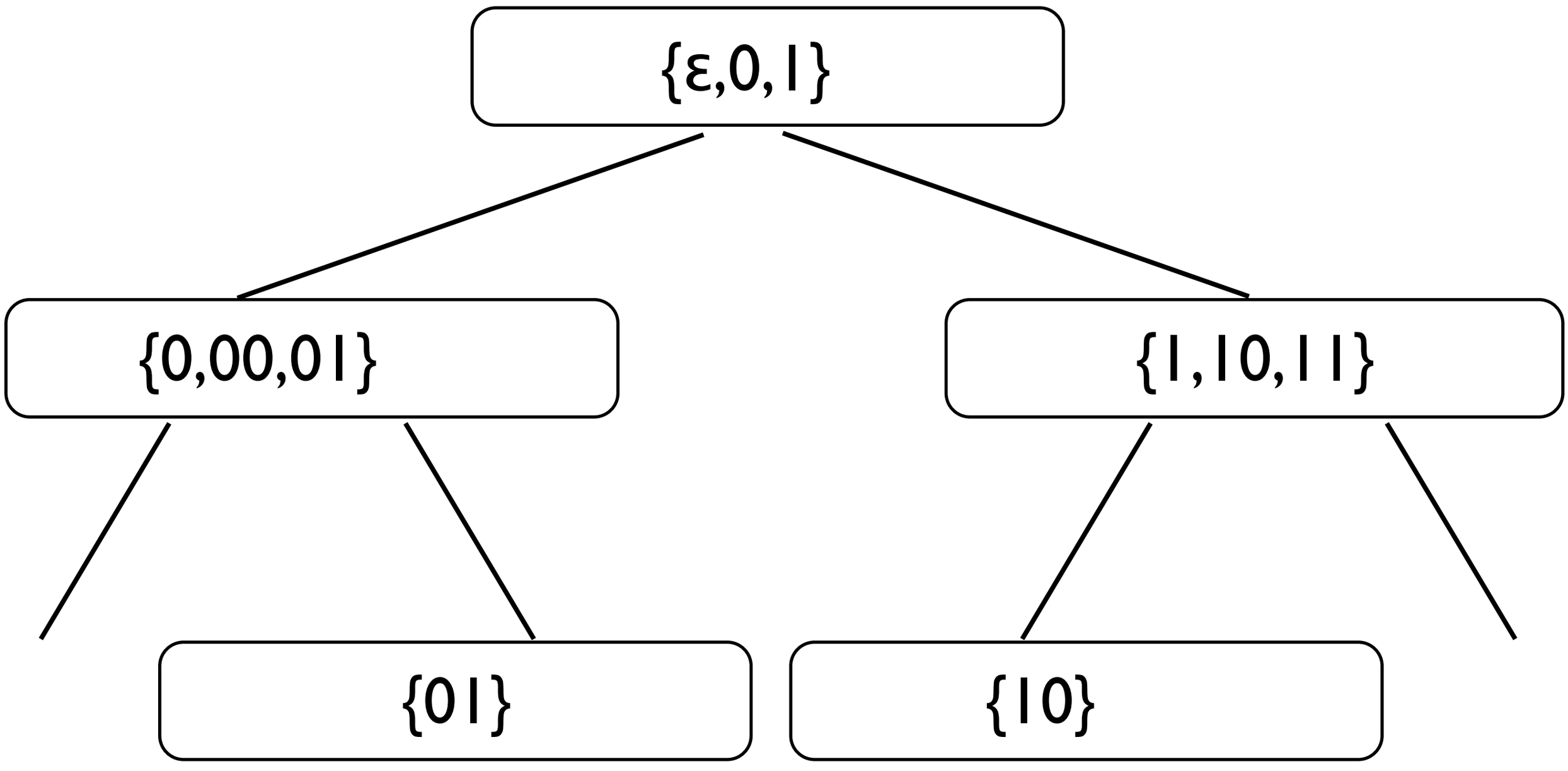
$\exists l_1, r_1, z_1\ .\ r_\varepsilon \rightarrow (l_1, r_1, nil) \wedge$
$ll_1 = ll_{10} \wedge z_1 = lr_{10} \wedge z_1 = ll_{11} \wedge lr_1 = lr_{11}$

$r_0 \rightarrow (nil, nil, lr_{01}) \wedge r_0 = ll_{01}$

$l_1 \rightarrow (nil, nil, lr_{10}) \wedge l_1 = ll_{10}$

Bounding the Tree Width

tll(root,ll,lr) ::= ∃l,r,z . root→(l,r,nil) * tll(l,ll,z) * tll(r,z,lr) | root→(nil,nil,lr) /\ root=ll



Bounding the Tree Width

$tll(root,ll,lr) ::= \exists l,r,z\ .\ root \rightarrow (l,r,nil) * tll(l,ll,z) * tll(r,z,lr) \mid root \rightarrow (nil,nil,lr) \wedge root=ll$

$\exists l_\varepsilon,r_\varepsilon,z_\varepsilon\ .\ root \rightarrow (l_\varepsilon,r_\varepsilon,nil) \wedge$
$l_\varepsilon = ll_0 \wedge \mathbf{z_\varepsilon = lr_0 \wedge z_\varepsilon = ll_1} \wedge lr_\varepsilon = lr$

$\exists l_0,r_0,z_0\ .\ l_\varepsilon \rightarrow (l_0,r_0,nil) \wedge$
$ll_0 = ll_{00} \wedge z_0 = lr_{00} \wedge z_0 = ll_{01} \wedge \mathbf{lr_0 = lr_{01}}$

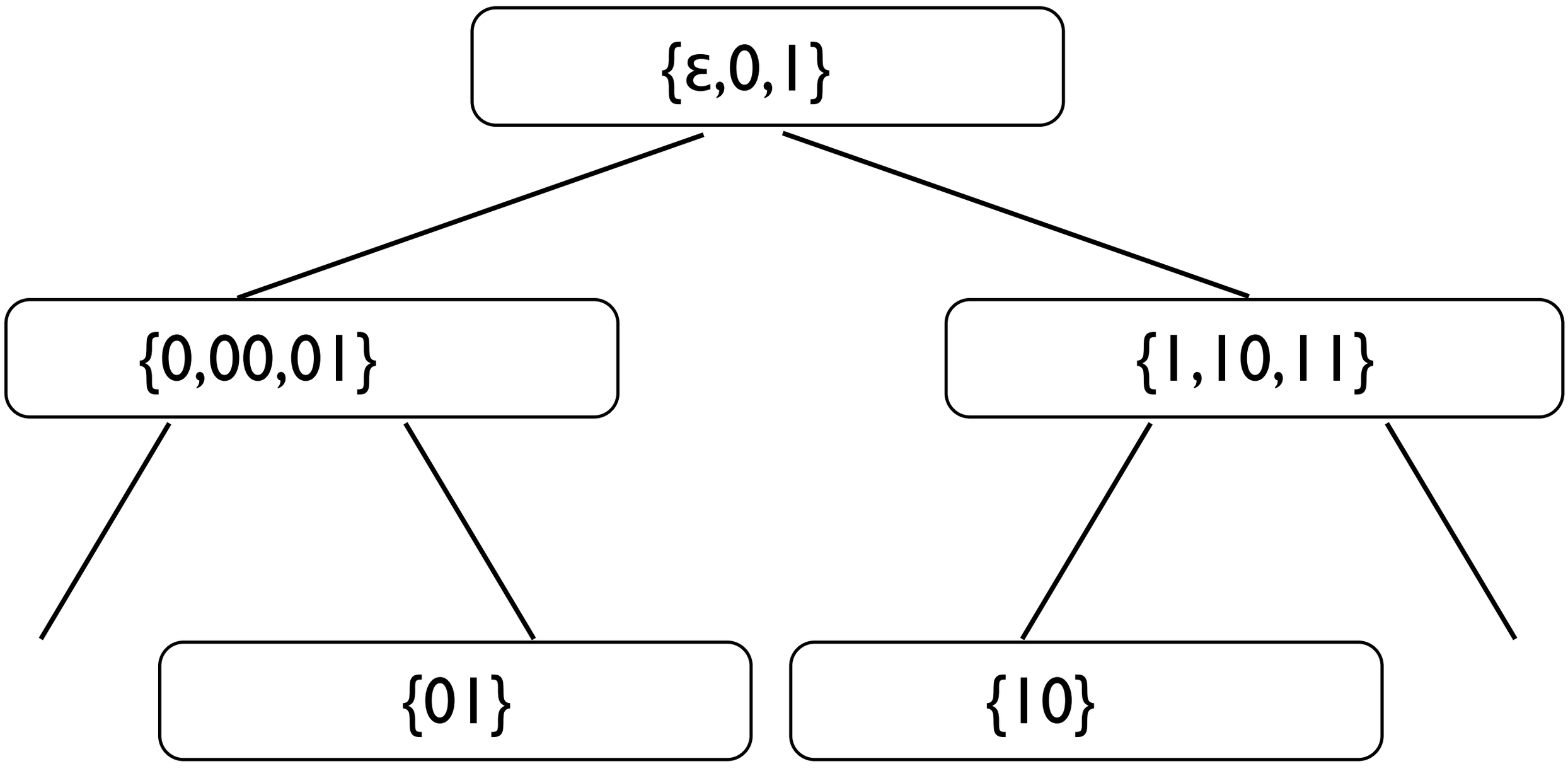
$\exists l_1,r_1,z_1\ .\ r_\varepsilon \rightarrow (l_1,r_1,nil) \wedge$
$\mathbf{ll_1 = ll_{10}} \wedge z_1 = lr_{10} \wedge z_1 = ll_{11} \wedge lr_1 = lr_{11}$

$\mathbf{ll_{01} \rightarrow (nil,nil,lr_{01})}$

$\mathbf{ll_{10} \rightarrow (nil,nil,lr_{10})}$

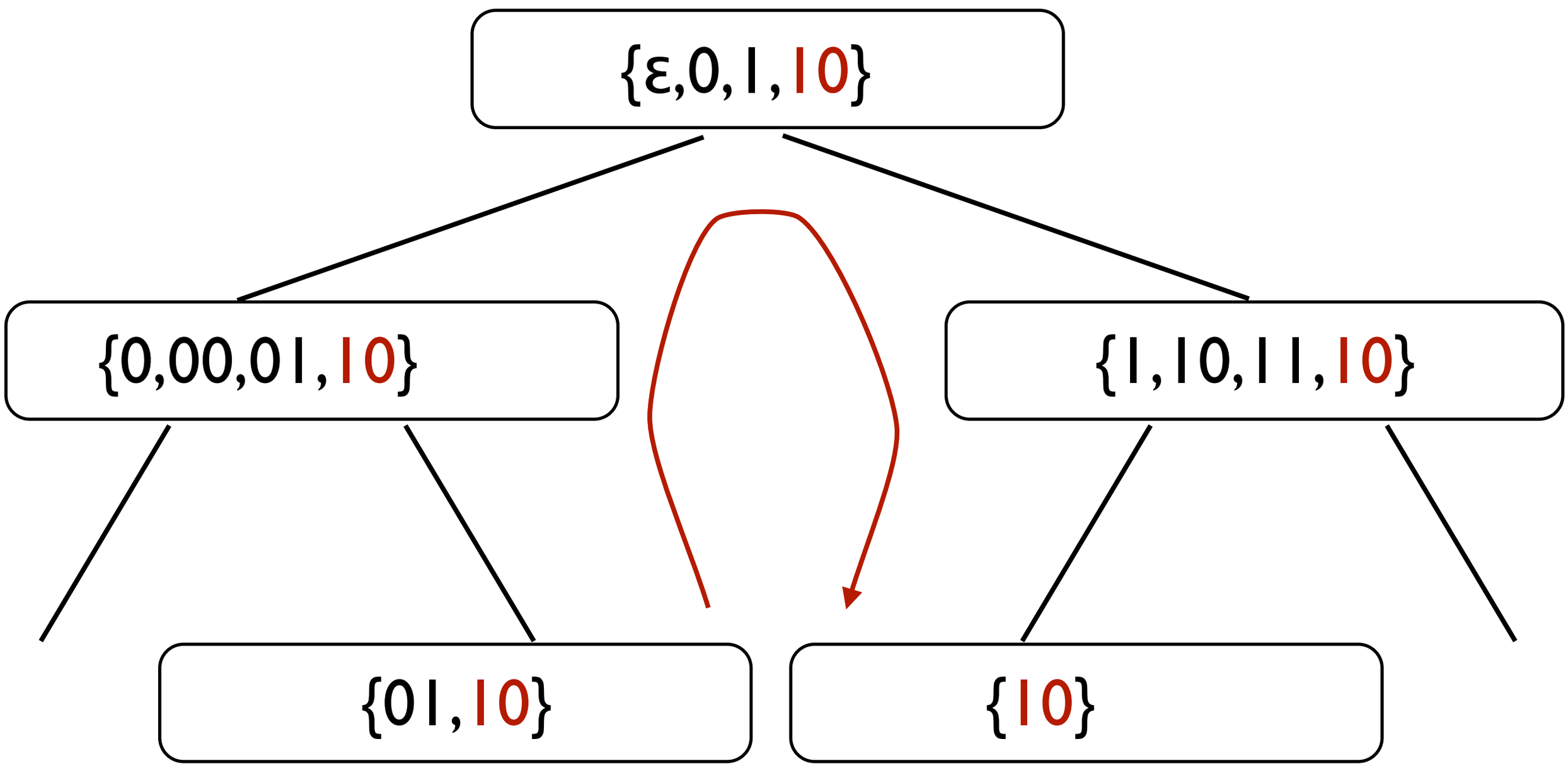
Bounding the Tree Width

tll(root,ll,lr) ::= ∃l,r,z . root→(l,r,nil) * tll(l,ll,z) * tll(r,z,lr) | root→(nil,nil,lr) /\ root=ll



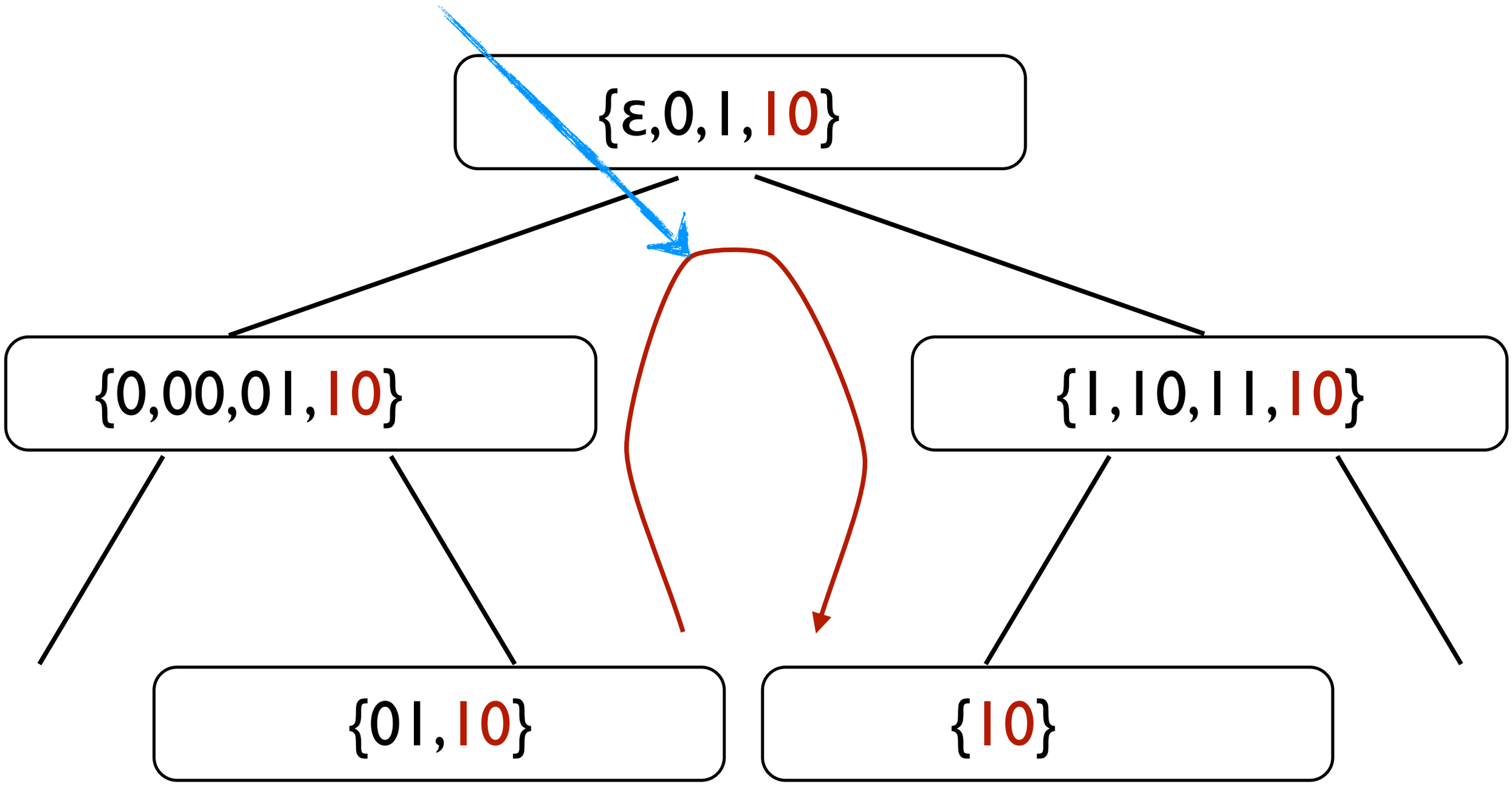
Bounding the Tree Width

tll(root,ll,lr) ::= ∃l,r,z . root→(l,r,nil) * tll(l,ll,z) * tll(r,z,lr) | root→(nil,nil,lr) /\ root=ll



Bounding the Tree Width

of non-local edges bounded by $||\{P_1 \dots P_k\}||$) /\ root=ll



Translating SLRD to MSO

1. Define the set of backbones:
    - trees containing all the allocated nodes and only some of the forward local edges
2. Define the rest of the edges using the backbone:
    - use a Tree Walking Automaton to track the destination of each edge starting from the source node [Bojanczik 2008]
    - encode the runs of the TWA in MSO and define the edges

Discussion on Complexity

- MSO on bounded tree width graphs is not elementary recursive [Meyer '75]
- Our translation to MSO generates formulae of fixed quantifier alternation (bounded by a constant)
- The translation to MSO on trees from Courcelle's Theorem only increases the alternation depth by a constant
- The complexity of SLRD entailments is elementary, and likely to be in EXPTIME

Conclusions

- Under some natural restrictions (connectivity, well-establishment) SL with Recursive Definitions is decidable (satisfiability and validity of entailments)
- All models are graphs of bounded tree width
- Translation from SLRD to MSO on graphs

Related Work

- Berdine, J., Calcagno, C., O'Hearn, P.W.: A decidable fragment of separation logic. In: Proc. of FSTTCS'04. LNCS, vol. 3328. Springer (2004)

- Cook, B., Haase, C., Ouaknine, J., Parkinson, M.J., Worrell, J.: Tractable reasoning in a fragment of separation logic. In: Proc. of CONCUR'11. LNCS, vol. 6901. Springer (2011)

- Brochenin, R., Demri, S., Lozes, E.: On the almighty wand. Inf. Comput. 211, 106–137 (2012)

- Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: Proc. of POPL'11 (2011)

- Møller, A., Schwartzbach, M.I.: The pointer assertion logic engine. In: Proc. of PLDI'01 (June 2001)